Evolution of web application frameworks

Currently just a braindump... feel free to improve - at least keep the positive attitude towards REST, please 😉

1. Webserver delivering static files
2. CGI - calling dynamic scripts, mapping configured by eg. regexps in static webserver
3. Templating languages allow meshing logic with Markup: PHP, ASP, simplifies the generation of HTML
4. Java servlets - fully Java interface to server-side HTTP, own "dynamic" webserver = servlet container
    - Base mapping with servlets: context path to separate servlets
    - Various mappings implemented in Java behind the servlet api (eg. Cocoon)
    - JSP as a template language for JavaServlets
5. Concentration of backends and server-to-server communication ("Enterprise" segment): J2EE with Java beans, RMI, JDBC, ORM
    - did not make any use of the advantages of REST present in HTTP and the web
6. Inter-server communication: webservices, XML-RPC, SOAP
    - frameworks with focus on automated generation of service endpoints
    - no HTML, no REST
    - no improvements for making it easier to quickly create nice user interfaces
    - tons of files (one resource = interface, bean implementation, dao object, etc., see here)
7. More standards: Java server faces, a full MVC object model inside the server
    - makes things more complicated, especially when considering a thin server architecture
8. Ruby on Rails: large improvement for the developer through scaffolding files, following the DRY principle
    - considered state-of-the-art
    - can be done with java as well: Grails
    - scaffolding (both through CLI scripts and method-not-found-interceptors in dynamic languages) make it easier to work with this large bunch of objects/classes/files on the server side
9. Sling: TNGWAF - the next generation web application framework
    - server-side made thinner through resources as a first-class concept
    - no need for complicated MVC patterns just to get from URL to relational database
    - flexibility through scripting
    - power of 10+ year development of Java VMs
    - OSGi for better software lifecycle management and more uptime (updates without restart)