Business-driven IT for SAP - The Model Information Flow

Guillaume Belrose, Klaus Brand, Nigel Edwards, Sven Graupner,
Jerry Rolia, Lawrence Wilcock
Enterprise Systems and Software Laboratory
HP Laboratories Palo Alto

HPL-2007-143
August 23, 2007*

business-driven IT, SAP, automated management, ITIL, ITSM, SOA

Enterprises rely on efficient and flexible IT services. While complexity of services is increasing, personnel to provide and manage services will remain limited. At the same time, IT environments are becoming more dynamic, from the business side as well as from the infrastructure side. The ability to incorporate change faster, more efficiently and reliably has become a measure of quality of enterprise IT organizations.

IT responds to these challenges by decoupling functions into services and by improving the linkages between business processes and the supporting IT systems. Service-oriented Architecture has become the accepted pattern for modern enterprise IT.

This paper presents the *Model Information Flow*. It is part of a collaboration between HP Labs and SAP Research. The goal of the collaboration is to explore new approaches of model-driven planning, design and management of enterprise applications in a shared and virtualized IT infrastructure. The goal is to substantially improve the linkage between the business and the IT layer and the ability to manage and accommodate change more efficiently and in a largely automated manner.

* Internal Accession Date Only
Published in BDIM 2007, 21 May 2007, Munich, Germany

Approved for External Publication

Business-driven IT for SAP – The Model Information Flow

Guillaume Belrose, Klaus Brand, Nigel Edwards, Sven Graupner, Jerry Rolia, Lawrence Wilcock

Hewlett-Packard Labs

Filton Road, Stoke Gifford, Bristol, UK and 1501 Page Mill Rd, Palo Alto, CA 94304, USA

Abstract — **Enterprises rely on efficient and flexible IT services. While complexity of services is increasing, personnel to provide and manage services will remain limited. At the same time, IT environments are becoming more dynamic, from the business side as well as from the infrastructure side. The ability to incorporate change faster, more efficiently and reliably has become a measure of quality of enterprise IT organizations.**

IT responds to these challenges by decoupling functions into services and by improving the linkages between business processes and the supporting IT systems. Service-oriented Architecture has become the accepted pattern for modern enterprise IT.

This paper presents the *Model Information Flow*. It is part of a collaboration between HP Labs and SAP Research. The goal of the collaboration is to explore new approaches of model-driven planning, design and management of enterprise applications in a shared and virtualized IT infrastructure. The goal is to substantially improve the linkage between the business and the IT layer and the ability to manage and accommodate change more efficiently and in a largely automated manner.

Keywords: business-driven IT; enterprise IT management; business to IT linkage; model-driven management; automated management; enterprise IT resource planning; IT design; service-oriented architecture; virtualization.

I. Introduction

While automation has made substantial progress on the business side of IT, such as in business process automation [1], automation in IT management has been lagging behind. On the business side of IT, enterprise software such SAP is widely been used to automate the business processes in enterprises. Tools such as Aris [2] are used to design automated business processes that are executed on the SAP platform. In IT management, in contrast, people still carry out management tasks and processes ranging from higher-ordered planning stages to the lowest levels of managing machines, networks and storage. Management tools are used that support those tasks. But the degree of automation is low.

Accommodating change has always been a challenge in IT. Change may originate from the business side such as changes to business processes when switching partners in a supply chain. Change may also originate from within IT such as when IT systems need to be maintained, updated or upgraded or need to comply with latest regulations.

Virtualization helps to decouple applications from resources enabling new opportunities to operate IT more effectively by consolidating applications, sharing resources and improving overall utilization. However, benefits emerging from virtualization also introduce another layer of management. In order to leverage advantages, virtualization itself must be managed effectively, ideally fully transparently and automatically like in an operating system.

Service-oriented Architecture (SOA) allows the decoupling of consumers and providers of IT functions by well-defined interfaces and open protocols. Formerly monolithic applications are broken apart such that they can be provided and consumed more flexibly as services. SOA can be applied at any layer, from businesses and organizations to processes; from applications to systems and resources. But the higher degree of modularity, choice and flexibility comes again to a price of increased effort in management. Services must be orchestrated and coordinated, eventually across domains when a service is provided in another than the consumer's domain.

With growing complexity, scale and connectivity of IT at all levels, management and in particular managing change has become a limiting factor for business efficiency in enterprises.

Model-driven approaches to IT management aim to improve and accelerate the design, management and change processes in IT. Models are used to formally represent information about IT systems at the various stages of their lifecycle, from planning and design to deployment, management, maintenance and final retirement. Availability of formal information as opposed to information informally carried by people can enable the use of advanced tools and the automation of design and management stages of IT systems. But despite the fact that models and model-driven processes have become prevalent in most modern industries, such as in chip or car manufacturing, model-driven approaches to IT management are at the beginning and remain subject to continued research.

SAP is a major provider of enterprise applications that need to be managed in IT. Hewlett-Packard is a major provider of IT infrastructure and management systems as well as services. Substantial effort is spent to plan, design, implement and manage SAP applications along with infrastructure throughout enterprises around the world. In a joint research collaboration between HP Labs and SAP Research, new approaches of model-driven planning, design and management are being developed and investigated. The goal is to link business processes better with the infrastructure supporting them. Models are used to reflect the different stages of requirements, designs, deployments and management environments. The ability to incorporate and evaluate consequences of change faster in models than in the real environment and driving actual changes in an IT environment automatically from models, is expected to deliver more efficient IT management processes and improved responsiveness of IT to change.

II. Problem Statement

The speed and cost of creating new applications in IT and incorporating change has become a major factor for IT for supporting the business of the enterprise. Experiences have shown that addressing only isolated domains of the problem space, such as infrastructure management or applications management, has limited effect. The main problem remains how the different processes, people and organizations that are involved at different stages can cooperate and exchange information more efficiently than they can today such that the entire chain from planning, design, implementation (including testing) to deployment, management and final retirement can be processed in a much better integrated manner based on formalized information that is being consumed and produced at each stage. Establishing such an information supply chain across the different stages in an integrated manner is one of the significant problems in enterprise IT today.

III. Approach

The novel aspect of our approach is the formalization, linkage and coherent interoperability of models and transformations along the several stages of business process customization to application and infrastructure design, and from there to the stages of deployment and management of a finally operating application. Models are conceptually and logically linked beginning from the business process and its customization, then leading to a choice of application components needed to support the customized processes, to the proper "sizing" of these application components in order to meet an anticipated workload. Information about sized application components then in turn provides requirements for infrastructure components and their configurations (servers, networks, storage) including eventual layers of virtualization.

Models are used to accurately capture the needed information at each stage. Transformations occur in order to derive a model of a subsequent stage from the information available at a prior stage. Transformations establish the linkages between the models. The linkages between models then allow incorporating change at potentially any stage and re-performing respective transformations only for the stages following the stage where a change had occurred. For example, in case that only infrastructure changes (e.g. caused by an equipment replacement cycle), the requirements from the business process and its breakdown into application components would remain unaffected. Sizing might change because newer equipment will likely have improved performance. The newly sized design then provides the information for the subsequent transformations into proper infrastructure configurations and application deployments. In case of a business process change, the entire chain of linked models may need to be recomputed and recreated.

IV. The Model Information Flow

The Model Information Flow represents a number of models that have been identified for capturing the information needed at the different stages. The Model Information Flow also represents a flow of information from "the left to the right". This flow of information exists today when enterprise applications are being planned, designed, implemented and managed. The difference to today's practice is that, in the Model Information Flow, formalized information flows in the form of models from the left to the right as opposed to informal information in the form of documents or other means of communication among people.

The information in the Model Information Flow includes:

- A general process pattern is chosen that matches the kind of process to be built. A customization steps follows.
- The customized business process then determines the application components that are needed to perform the transactions used by the process.
- Non-functional requirements are taken into account such as performance, security and availability. Application sizing is an established step in enterprise application design which determines the capacity or the amount of resources needed for the anticipated workload.
- Functional requirements (business processes and process steps) and non-functional requirements (performance, security and availability) then provide the input for an initial application design including the actual application components with numbers or ranges of instances.
- Infrastructure must then be chosen and configured such that it can support the application design. While in the past this step was largely a matter of choosing resources (networks, machines, storage), this is changing and becoming more and more a matter of exploring configuration choices and virtually creating the needed resources in shared IT environments. This means, resources need to be allocated from pools and virtual resources (networks, machines, storage) may need to be created in them. Exploring infrastructure design choices based on application requirements and reflecting those choices in form of infrastructure designs is becoming more and more relevant. An infrastructure design is the result that is tailored for the requested application design in the targeted data center environment.
- Once an infrastructure design has been created, it can be instantiated by applying configuration parameters from the design to the physical environment in the data center.
- Once the resource infrastructure has been brought into existence, the application design can be deployed and configured in the desired form as specified.
- Once the application is deployed, its management lifecycle becomes effective through which then both, the infrastructure and the application are managed.

By formalizing this information, some of the transformations can be automated and hence accelerated such that they can be performed and re-performed faster. Formalization also allows exploring larger design spaces using techniques such as Layered Queuing Models [3], policy-based design [4] or genetic algorithms [5].

Figure 1 gives an overview of the identified models and transformation for the Model Information Flow.

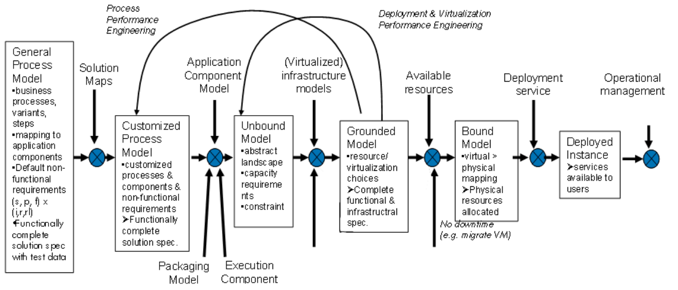
Process Performance Engineering
Deployment & Virtualization Performance Engineering
General Process Model
•business processes, variants, steps
•mapping to application components
•Default non-functional requirements (s, p, f) × (i,r,rl)
Functionally complete solution spec with test data
Solution Maps
Customized Process Model
•customized processes & components & non-functional requirements
➢Functionally complete solution spec.
Application Component Model
Packaging Model
Execution Component
Unbound Model
•abstract landscape
•capacity requirements
•constraint
(Virtualized) infrastructure models
Grounded Model
•resource/ virtualization choices
➢Complete functional & infrastructral spec.
Available resources
Bound Model
•virtual > physical mapping
➢Physical resources allocated
No downtime (e.g. migrate VM)
Deployment service
Deployed Instance
➢services available to users
Operational management

Figure 1: Model Information Flow with transformations.

The following section explains the individual, constituent models of the Model Information Flow in more detail.

V. Constituent Models

The following models have been identified for the stages of the Model Information Flow:

- The General Process Model.
- The Customized Process Model.
- The Application Packaging Model.
- The Constraints Model.
- The Application Performance Model.
- The Unbound Model.
- The Infrastructure Capability Model.
- The Grounded Model Design.
- The Grounded Model.
- The Bound Model.
- The Deployed Model.

A. *The General Process Model*

As General Model we understand a number of blueprints that exist for general SAP environments that can be reused and customized. The General Process Model represents the standard business process blueprint that is delivered with SAP for a number of standard business processes for a number of business domains such as Supply Chain Management (SCM), Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), Product Lifecycle Management (PLM) or Supplier Relationship Management (SRM) as well as industry segments such as Financial Services, Public Services, Service Industries, Process Industries, Consumer Industries or Discrete Industries [6]. The spectrum and classification of these domains and industries is defined by SAP in the Enterprise Service Architecture (ESA) [7]. A large library of those blueprints exists in SAP/R3 Solution Manager. Work is organized in projects. Steps to define projects and processes are documented in [8].

Selection of a General Process Model from the library of blueprints is the first step of designing a business process. This blueprint is then subject of customization leading to the Customized Process Model.

B. *The Customized Process Model*

The Customized Process Model represents the customization of a business process for a project. The business process consists of a number of steps that also can be hierarchical. Tools such as Aris [2] can be used to define and customize processes.

A customized business process represents a sequence of activities that is initiated in a client, which can be a user in front of a terminal or another program. Steps of a business process may lead to transactions in the associated SAP/R3 server. Steps in a process that only describe user activity on the terminal, such as filling in information, do not cause transactions and are not relevant for the server.

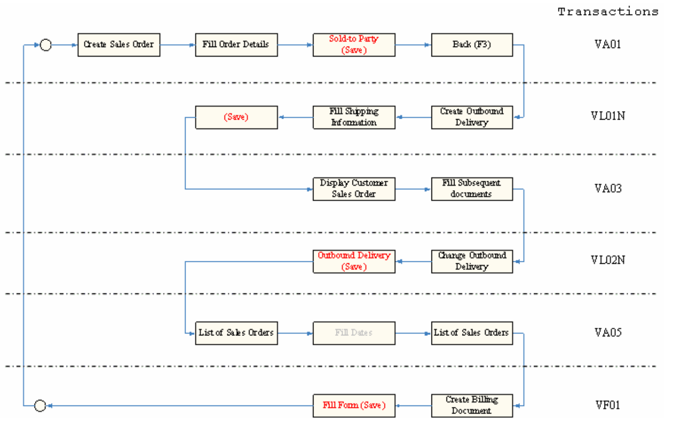

Figure 2: Customized process for Sales and Distribution (SD).

Figure 2 shows the customized process for the Sales and Distribution (SD) process, which is often used as a benchmark for SAP applications. The SD process consists of 17 steps that are executed as a sequence by a user. Steps in the right-most column cause transactions on the SAP/R3 server. Transactions used in the SD process are: VA01, VL01N, VA03, VL02N, VA05 and VF01.

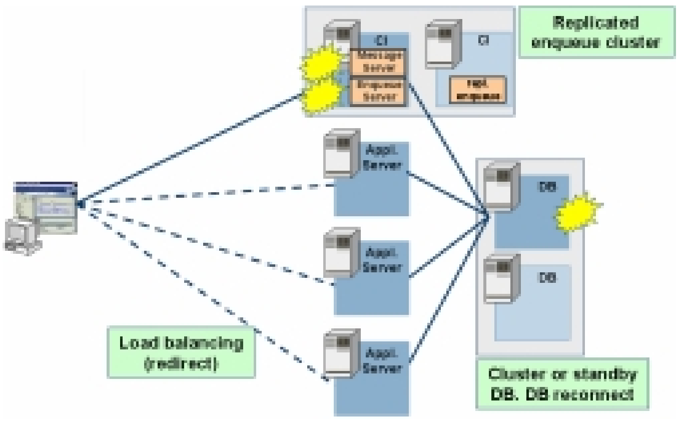
CI
Message Server
Enqueue Server
CI
repl. enqueue
Replicated enqueue cluster
Appl. Server
Appl. Server
Appl. Server
DB
DB
Load balancing (redirect)
Cluster or standby DB. DB reconnect

Figure 3: Three-tier client-server architecture of SAP/R3.

Figure 3 shows the client-server architecture of a typical SAP/R3 system where users execute dialog steps as defined by the customized business process which lead to transactions on the SAP server, which in turn cause transactions on the database. While steps are executed on the client side (by a user or another program), transactions caused by steps are executed on the (SAP/R3) server side in application components. Each application component typically offers one type of transaction. Multiple steps may invoke the same transaction type, and hence the same application component.

While the Customized Process Model primarily defines the functional requirements of the process, it also includes non-functional requirements such as requirements for performance, availability or security that are required for that process. Performance requirements, for instance, describe how many clients or users are expected to use the system executing the customized process. The number of expected concurrent users is an important measure of the capacity for which a SAP system will be sized.

Figure 4 shows the formal representation of the Customized Process Model in the Model Information Flow as UML class diagram. The model shows business processes and their steps and how they relate to application components, particularly which application components will execute business process steps, and which performance requirements will be imposed on components.

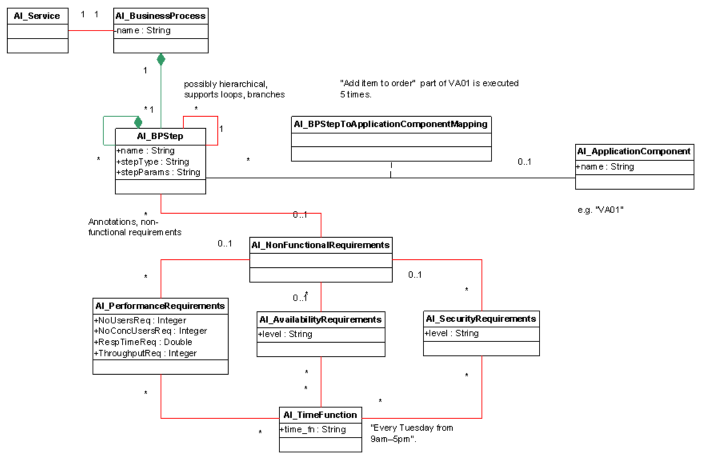

Figure 4: Formal representation of the Customized Process Model as UML class diagram.

The model shows a one to one correspondence between an instance of an AI_Service and an AI_BusinessProcess. The AI_Service is the information service that implements the business process. It is the general anchor class for the entire Model Information Flow model chain.

A business process can be decomposed into a number of business process steps. Instances of the AI_BusinessProcess class contain one or more AI_BPSteps. An instance of an AI_BPStep may be broken up into multiple smaller AI_BPSteps when involving branches or loops. Once a business process step is decomposed into sufficient detail each of the lowest level steps can be associated with an AI_ApplicationComponent that will execute the step. An application component is the program or function that implements the execution of a business process step such as the SAP transaction named VA01 in the SD process.

The relation AI_BPStepToApplicationComponentMapping is a mapping that details how the business process step is mapped to the application component. It provides the linkage between specific steps to the transactions invoked within the application component. It also provides details of parameters, such as the average number of line items in a sales order, etc. Not all steps require interaction with application components such as certain interactions with a user that control the flow of steps (e.g. choices a user makes in the UI). In those cases, no transactions are performed on SAP application components and hence no association exists between this step and an application component.

A business process step may have a set of non-functional requirements (class AI_NonFunctionalRequirements) associated with it that are imposed on application components when steps are executed. Requirements include: performance, availability and security requirements. In the current version, availability and security requirements are modeled as a string holding an expression such as, in the simplest case, “high”, “medium”, “low”. Performance requirements are specified in terms of numbers of registered users (NoUsersReq), numbers of concurrent users of the system, the expected response time in seconds and a throughput requirement for the number of transactions per second. Many steps may share the same set of non-functional requirements by aggregating them to instances of AI_NonFunctionalRequirements. A time function is also denoted by a string expression in class AI_TimeFunction. Instances of that class can be attached to individual instances of performance, availability or security requirements. These specify when the non-functional requirements apply, so different requirements can apply during office hours or outside normal office hours. Richer time varying functions are also possible to capture end of months peaks and the like.

In summary, the Customized Process Model specifies a customized business process consisting of steps that each may carry different non-functional requirements that are imposed on the transactions each step poses onto the server(s) containing the application components that are actually executing the transactions. This fine-grained modeling allows a detailed capture of business requirements in combination with non-functional requirements.

C. The Application Packaging Model

Figure 5 shows Application Packaging Model, which further expands the Customized Process Model starting with an AI_ApplicationComponent. The Application Packaging Model describes the internal structure of the software: what products are needed and what modules are required from which products. The model describes that an AI_Application-Component is associated with an AI_ApplicationModule. An application module might correspond to a JAR file for an application server. In the case of SAP/R3, it might be the module to be loaded from a specific product into an application server such as SD or FI.

One or more application modules can be included in a software product. For example, the SAP/R3 Enterprise product contains the modules for SD. Application modules can be dependent on other application modules. For example, the SD code module for the application server depends on the SD data being loaded into the database.

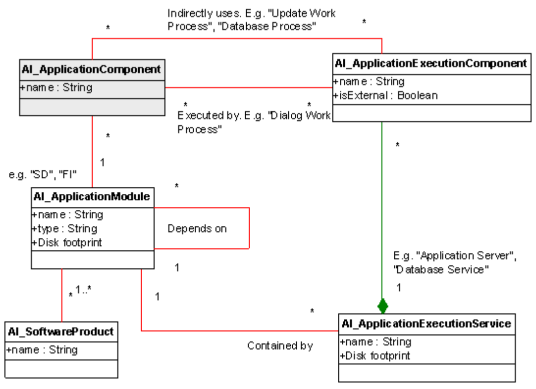

Figure 5: The Application Packaging Model.

An application component is executed by an AI_Application-ExecutionComponent. This could be a servlet running in an application server or a web server. It could also be a process such as a Unix process. In the case of SD's VA01 transaction, it is a Dialog Work Process that is executing the component. When it executes, the application component may indirectly use or invoke other application components in order to run. SD transactions need to access other application components as well such as the Enqueue Work Process and the Update Work Process, as well as the database application execution component. This is why there are two relations between AI_ApplicationComponent and AI_ApplicationExecution-Component. The "indirectly uses" association shows which additional application execution components might be used whenever an application component is executed. This is useful for determining the components that need to be installed in order to produce a working service.

To summarize, the Application Packaging Model contains the information in which application modules application components are included and where they will be executed. An application execution service (such as a SAP application server) loads or contains application modules (such as SD) that execute in application execution components (such as Dialog WP) which, in turn, execute the application component (such as VA01) in order to deliver a business process step.

Note, that the Application Packaging Model describes the topology as part of the requirements for the service to be created. It itself is a design rather than a current status of the environment.

D. The Constraints Model

Constraints are needed by transformations and tools or people carrying out transformations to drive models from one step in the Model Information Flow to the next. There is a need to express arbitrary constraints on classes and instances for each of the various models. Constraints can be "soft" constraints such as preferences, guidelines or hints or can express "hard" constraints such as impossible or undesired combinations.

Policy-based design has been explored in the past for modeling valid or preferred hardware configurations and driving configuration generation processes solely from constraints [4]. Constraints were included as strings in modeled classes using a constraint language [9]. The problem with the approach was that constraints turned out to be case and even for the same case time specific, while models were supposed to be reusable over time as well as across deployments for similar configurations.

From this experience we learned to not include constraint expressions directly into models, rather make them attachable to elements in models (classes and instances). This is the approach taken in the Model Information Flow. Constraints expressions are factored out from being contained in model classes. They are described as separate instance data that can be attached to (and potentially be shared with) any instance or any class in the other Model Information Flow models.

Constraints are expressed as instances of class AI_Constraint, as shown in Figure 6, which is capable of holding arbitrary constraint expressions as strings.

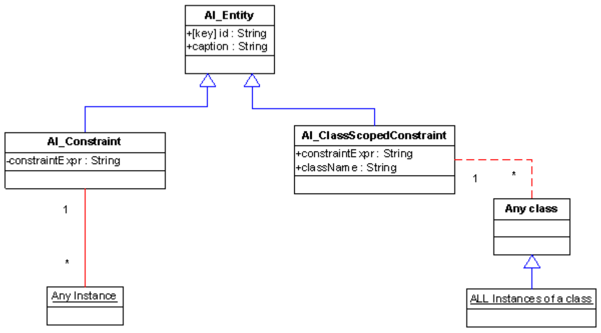

Figure 6: The Constraints Model.

The class diagram Figure 6 shows that there are two constraint classes, one for so-called class-scoped constraints and one for instance-scoped constrains. Instances of those classes can hold constraint expressions that can be attached to either any other instance or any class to which they are attached. Constraints may also apply to associations.

Constraints are used by tools to generate new models and model information as the Model Information Flow progresses from left to right. Examples of constraints include:

- How to scale up application servers – which application execution components are replicated and which are not?
- Installation and configuration information for application components, application execution components and application execution services.
- Performance constraints on application execution services such as avoid running an application server on a machine with greater than 60% CPU utilization.

E. The Application Performance Model

Figure 7 shows the class diagram of the Application Performance Model. The purpose of this model is to define the resource demands required for business process steps (as direct demands) and for demands application components impose on other application components (as indirect demands) in effect of executing transactions.

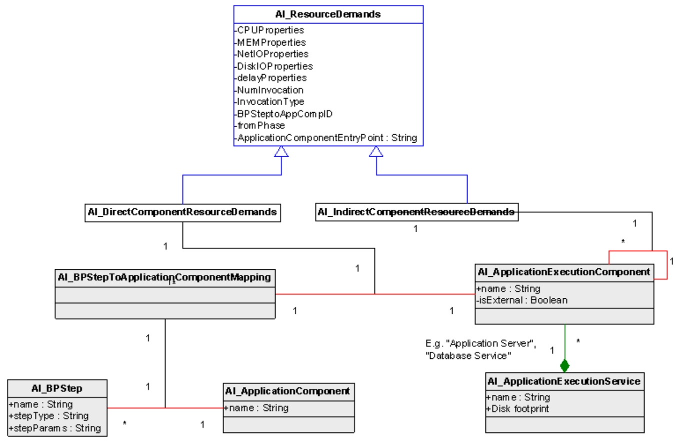

Figure 7: The Application Performance Model.

There are two types of resource demand to consider.

1. The demand for resources generated directly by the application execution component (e.g. a Dialog WP) using CPU, storage I/O, network I/O, memory and other metrics when it executes a business process step – this demand is modelled by class AI_DirectComponent-ResourceDemand.
2. The demand for resources generated by components that the above application execution component uses when it calls or invokes other components (e.g. a Dialog WP using an Update WP) – this demand is modelled by class AI_IndirectComponentResourceDemand.

AI_IndirectComponentResourceDemand can be recursive (e.g. the UpdateWP might invoke a database process), which means there are further instances of that class between further instances of AI_ApplicationComponents.

The following is explanatory text for some of the properties that appear in AI_IndirectComponentResourceDemands and AI_DirectComponentResourceDemands. The properties are inherited from the common superclass AI_ResourceDemands.

CPUProperties may be expressed in a higher-ordered measure such as SAPs [11]. Similar measures can be used for expressing *MEMProperties*, *NetIOProperties* and *DiskIO-Properties*. *delayProperties* allow to express any delay (e.g. a wait or sleep) associated with the component's activity which does not consume any CPU, NetIO and DiskIO resources. *NbInvocation* allows expressing the number of times the component is invoked during the execution of a business process step. *InvocationType* indicates whether an invocation is synchronous if the caller is blocked or asynchronous if the caller can immediately continue.

The AI_DirectComponentResourceDemands and AI_Indirect-ComponentResourceDemands associations specify the unique resource demands for each business process step in a fine-grained manner. These demands properties are determined from known characteristics of each application component. They can be derived from known benchmarks and also from traces of installed systems.

To summarize, the Component Performance Model allows describing known performance characteristics for each application component directly and indirectly that is invoked in effect of a business process step. Information in this model requires detailed knowledge about the performance behaviour of the process and its application components. Part of this information is available today for SAP in form of well-known benchmarks. The remaining information needs either be collected from existing deployments or can be estimated by simulation such by the Layered Queuing Model tool [3] we use for the Model Information Flow.

F. The Unbound Model

The Unbound Model conceptually aggregates the discussed models: the Customized Process, the Application Packaging, the Constraints and the Application Performance Model. No new information is introduced that is not already contained in one of these models. Figure 8 conceptually shows the Unbound Model.

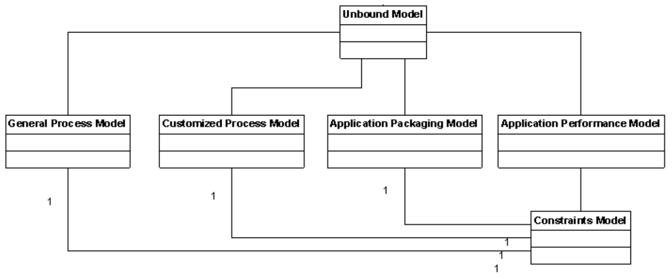

Figure 8: The Unbound Model.

The Unbound Model represents a marking point in the Model Information Flow. It aggregates the mentioned models that have in common that they represent requirements for an enterprise application, from the business process to its break down in application components and application execution services that are required to execute them. It is also used to determine the resource demands on those components from performance requirements according to which application execution components then need to be sized (which occurs in the step to the Grounded Model Design, which is explained in section H).

Requirements from the Unbound Model next need to be matched with resources that need to be made available in an infrastructure in order to deploy and run the application.

Since resource infrastructure itself has become configurable and even may need to be created based on a specification, a design of such a resource infrastructure is created first. Creation of resources as part of the later deployment process has become a major obstacle for traditional management processes and systems when dealing with virtualized resources that explicitly need to be created before they can be used.

For creating an appropriate resource infrastructure, a spectrum of choices exists that needs to be explored in order to identify a "good" match between the requirements from the Unbound Model and the capabilities resource infrastructure offers.

It means that the resources are not simply allocated from an existing inventory. The desired resource environment must be designed according to the requirements expressed in the Unbound Model for an application. Introducing this intermediate step of creating a resource infrastructure design first before the resources from that design are actually chosen from resource pools in a data center for deployment is an essential development introduced in the Model Information Flow. It allows decoupling the matchmaking between requirements to exploring choices of resource configurations from actual resource assignments in a data center.

There are two stages of resource infrastructure designs, one is called the Grounded Model Design, which is explained in section H and the following stage is called the Grounded Model, which is subsequentially explained in section I.

G. *The Infrastructure Capability Model*

Making requirements with capabilities is an inherently complex process. Several approaches have been explored in the past for assigning and allocating resources to requirements. There is extensive literature on the topic [12], [13], [14]. Some approaches applied complex optimization techniques; others used simpler bin packing. Most were restricted to singular resource types, which is insufficient in practical environments. Alternatively, policy-based resource topology design approaches have also been explored [4].

The approach taken in the Model Information Flow is simpler. It is based on the idea to describe (enumerate) a finite number of possible resource infrastructure configurations in a catalog of so-called Infrastructure Capability Models, from which possible resource configurations can be chosen and further parameterized for final deployment. The catalog contains a number of instances of Infrastructure Capability Models, which are defined in Figure 10. Capabilities may vary from data center to data center. Using a catalog from which resource configurations can be chosen as capabilities simplifies the matchmaking process to application requirements from the Unbound Model.

Figure 9 shows the abstract transformation how a resource design is derived from application requirements summarized in the Unbound Model by choosing and parameterizing a resource configuration chosen from the catalog of Infrastructure Capability Models.

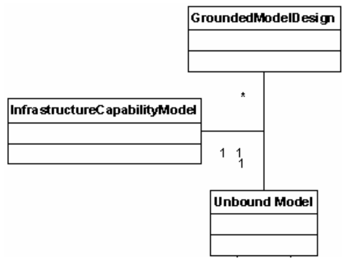

Figure 9: Deriving a resource design from requirements in the Unbound Model using a catalog of Infrastructure Capability Models.

Presenting a finite catalogue of resources that can be instantiated leads to a finite number of choices. This makes the selection of resource types by a capacity planning tool simpler [15]. It also makes the infrastructure management easier as there is less complexity in resource configuration. Standard templates can be used.

Another decision that has been made was to not expose the hosting relationship for virtualized resources. The DMTF Virtualization System Profile [10] models hosting relationship as a "HostedDependency" association. This also keeps the models simpler since it avoids dealing with recursion.

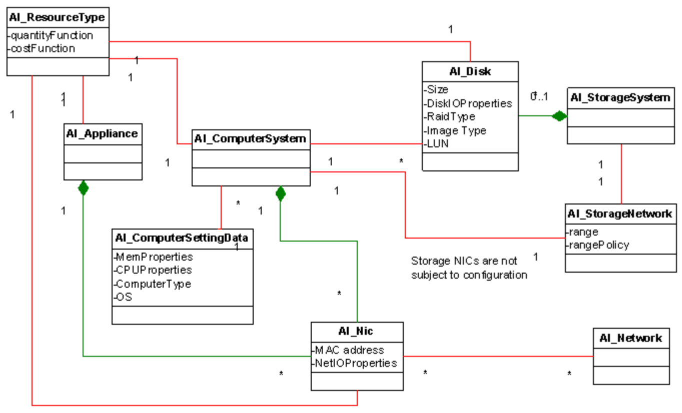

Figure 10: The Infrastructure Capability Model.

The Infrastructure Capability Models is defined as class diagram in Figure 10. It contains classes for resource types such as AI_ComputerSystem or AI_Device that can be deployed and configured by the underlying resource fabric (which is a management system in a modern data center that allows automatically configuring and deploying resources based on a formal specification).

H. *The Grounded Model Design*

As mentioned, the Grounded Model Design is an intermediate stage in the Model Information Flow to represent an abstract resource design that matches the requirements for an application summarized in the Unbound Model. Figure 11 shows the class diagram for the Grounded Model Design.

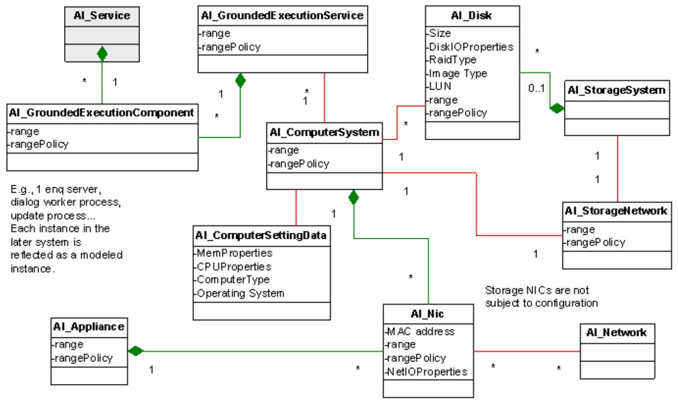

Figure 11: The Grounded Model Design.

Characteristics of the Grounded Model Design are:

- There is an instance of an AI_GroundedExecution-Component for each unique instance of an AI_-ApplicationExecutionComponent in the Unbound Model.

- One or more AI_GroundedExecutionComponents are executed by an AI_GroundedExecutionService. The execution association is consistent with that expressed in the Application Packaging Model.
- One or more AI_GroundedExecutionServices are run on an AI_ComputerSystem whose type has been selected from the Infrastructure Capability Model.
- A range attribute indicates the maximum and minimum number of components that might be used in the Grounded Model. Several different Grounded Models with different numbers of components may be derived from one Grounded Model Design.
- A rangePolicy attribute specifies how the appropriate number of components is selected from a range. This might be derived or influenced by the time varying AI_NonFunctionalRequirements in the Customized Process Model. It can be used by an infrastructure management system to determine when and under what conditions to switch between different Grounded Models.
- If the range and rangePolicy attributes are not set in an element within the Grounded Model Design, then there can be only one instance of the element of that type in any corresponding Grounded Model.

The Grounded Model Design is calculated from the Unbound Model using the catalogue of Infrastructure Capability Models. The total capacity of the system must satisfy the time varying performance requirements in the Customized Process Model. The required capacity is determined by combining these performance requirements with the aggregated resource demands (direct and indirect) from the Application Performance Model.

I. *The Grounded Model*

The Grounded Model is then the specification of a concrete resource infrastructure and the applications to be deployed on that infrastructure.

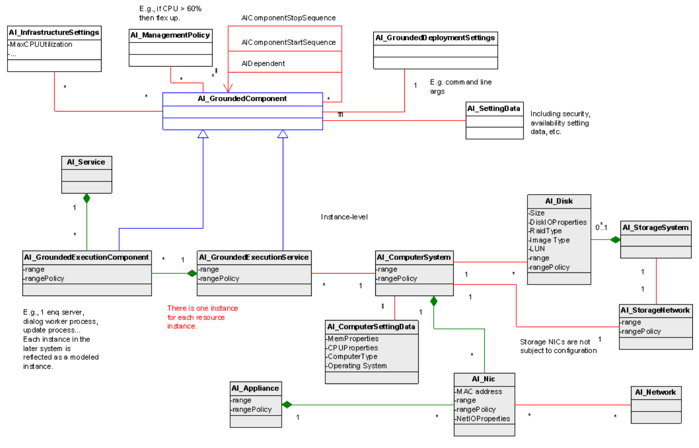

Figure 12: The Grounded Model.

Figure 12 shows the class diagram for the Grounded Model. Classes that have already been introduced in earlier models are shown in grey. It is apparent that it is very similar to the Grounded Model Design. The main new element is the super class AI_GroundedComponent. It was introduced to represent the installation and configuration information for both grounded execution components and grounded execution services, as well as information about policies and start/stop dependencies. Further properties define:

- *AI_InfrastructureSettings* contains threshold information for the management components, for example MaxCPU-Utilization – if it rises above a set figure such as 60%, an alarm should be triggered.
- *AI_ManagementPolicy* specifies further information for the management components – e.g. flex up if utilization rises above 60%.
- *AI_GroundedDeploymentSettings* include command line and configuration information so that the system can be installed, configured and started in a fully functional state.
- *AI_SettingData* provides additional configuration information that can override information provided in the grounded deployment settings. This allows grounded components to share the same set of deployment settings.

Not all attributes are set in the Grounded Model. For example, it is not possible yet to set MAC addresses in the Grounded Model, since there is not yet any physical resource assigned.

J. *The Bound Model*

The class diagram for the Bound Model is shown in Figure 13.

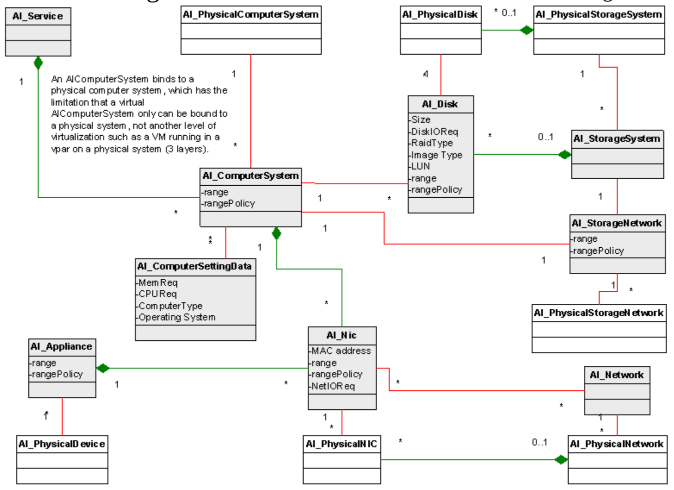

Figure 13: The Bound Model.

The Bound Model represents the binding of the Grounded Model to physical resources. It adds associations between the classes of AI_ComputerSystem, AI_Disk, AI_StorageSystem, AI_Network, and AI_NIC that are specified in the Grounded Model to real physical components that are available in the data center and that have been chosen for the application for deployment. A deployment system such as SmartFrog [15] in the prototype then deploys the bound model.

K. *The Deployed Model*

The Deployed Model differs from the Bound Model in only one respect. It shows the binding information for the management services running in the system. All the entities shown in this class diagram have a management service associated, which can be used to change state and/or observe the current state. One example of this could be to manage a virtual machine migration. The application managing the migration would use the management service in order to perform the migration. Once the migration is completed, the management application would update the Deployed Model and the Bound Model to reflect the new physical system.

Figure 14 shows the class diagram of the Deployed Model.

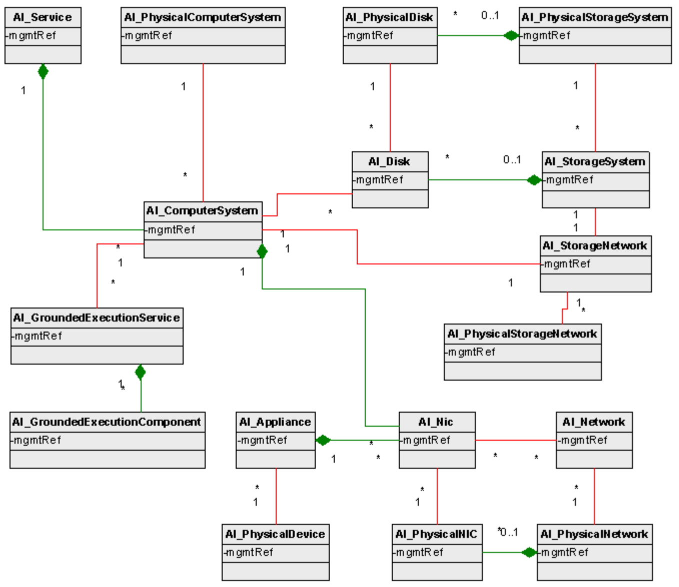

Figure 14: The Deployed Model.

VI. MODEL TRANSFORMATIONS

As information flows through the Model Information Flow, transformations occur between the various stages. These transformations occur in today's practice, but are solely manual and typically performed by different teams. Business consultants work out business processes in an enterprise. Solution architects then design and size an application based on the consultants' input. Integration teams build, test and integrate the application into a customer's data center.

The different transformations in the Model Information Flow occur at three different time scales: at design, deployment and run-time. Transformations are complex in nature and may not be automatable. Even in those cases, tools can support the exploration of design spaces, providing solution architects with better information for making decisions.

The approach taken in the Model Information Flow is to initially reflect today's practice by gathering information and describing it in models. Most transformations are initially performed manually before they can be successively supplemented and eventually replaced by tools.

A. Transformation: General Process Model to the Customized Process Model.

This first transformation is made by a consultant who chooses a business process blueprint from SAP's library [6] matching a customer's business case. The blueprint is then customized to the particular case. The result is a customized business process describing the steps a user will perform when exercising the process. Transactions are identified for steps. These describe the functional aspects in the Customized Process Model. Non-functional requirements need to be included such as the expected number of concurrent users of the process. Additional non-functional requirements can be captured as well such as for availability and/or security. Those requirements are included as text in the model, which could be informal text for human interpretation at later transformation stages, or could be expressed in a specification language for eventual subsequent transformation tools.

B. Formulating the Unbound Model

Based on the Customized Process Model, two other models need to be formulated.

The Application Packaging Model further details the transactions from the Customized Process Model into application modules and software packages based on information provided by SAP. Application execution components (e.g. work processes) and application execution services (e.g. application servers) based on which the application will run later need to be identified (see Figure 5).

The Application Performance Model then contains detailed descriptions of resource demands transactions pose on application components as well as among application components themselves when they invoke each other (see Figure 7). This detailed information about resource demands may be obtained from detailed measurements in reference installations. Estimates can also be used. It is one of the research questions to what extent fine-grained resource demands for transactions can be obtained in practice.

In addition, models can be supplemented with additional constraints from the Constraints Model indicating preferences or exclusions.

The entirety of General and Customized Process, Application Packaging and Performance Models forms what is called the Unbound Model. The Unbound Model summarizes all requirements of an application design and provides the input for the following transformation stage. The Unbound Model reflects the application view independently of the infrastructure in which it may be deployed. This independence allows for reuse of the Unbound Model. The following transformation links it to a particular target environment.

C. Transformation: Unbound Model to the Grounded Model

The transformation from the Unbound to the Grounded Model is the most complex transformation and a two-stage process. First, a Grounded Model Design is obtained, which is then refined into the Grounded Model, which contains descriptions of all the resources to which it can be bound and deployed.

Obtaining the Grounded Model Design as a first step means determining a set of resource types for all the application execution services that occur in the Application Packaging Model. These resource types need to be matched against the capabilities offered by the infrastructure of a data center. For example, if an Oracle 10 database (as type) is required in the

Application Packaging Model, and the list of infrastructure capabilities enumerates availability of Oracle 10 for HPUX on a number of potential server platforms as well as on Linux and Windows, all these capabilities will be taken into account. Each combination of operating system and server platform with availability of Oracle 10 also provides an estimate of the capacity the platform will deliver as well as cost (e.g. reflecting license cost). Both are included as properties in the Infrastructure Capability Model (refer to section G).

Another dimension of choice comes into play in form of the “size” of the platform to meet the performance requirements from the Customized Process Model. In order to obtain a database configuration of a proper capacity, certain server platforms may be insufficient and will be excluded from the list of valid choices. The number of instances may vary in case of a clustered database. Detailed resource demands from the Application Performance Model can be used to estimate performance by simulation, e.g. by using Layered Queuing Models [3]. For each platform choice, the number of instances can be varied in order to find a balanced design. A final choice must be determined by judgment of a solution architect based on the choices evaluated and presented as result of the design space exploration.

The Grounded Model Design then appears as a set of valid platform choices for all application execution services from the Unbound Model. For flexible deployments where resources can be scaled up and down, ranges of resources are determined within which the design remains balanced. From there, initial numbers of resources are determined which are then unfolded in multiple resource instance entities in the Grounded Model that represent the final design that is ready for binding and deployment.

D. Transformation: Grounded Model to Bound Model

While prior transformations occurred at design time, the transformation into the Bound Model occurs right before deployment. It means the assignment of resources from a data center to the resource instances represented in the Grounded Model. The inventory of available resources needs to be consulted in order to identify the available resources. Resource assignment is still a logical step meaning that resources are reserved for the application and cannot be reassigned elsewhere. Actual configuration and use of resources for the application occurs in the following step of deployment.

E. Transformation: Bound Model to Deployed Model

Once resource assignment has been completed, the resources are known and ready for use. Deployment of the application onto those resources can be carried out by a deployment service. Management service endpoints are created and their references are inserted into the Bound Model turning it into the Deployed Model.

VII. SUMMARY

The Model Information Flow establishes a linked chain of formal models reflecting all necessary information needed for defining, deploying and managing a SAP application. The initial approach taken was to identify and reuse the information people create and use when designing, building and integrating SAP applications. A set of transformations has been identified which create new information from existing information. Transformations in today’s practice are solely carried out manually. By introducing formal definitions in the form of models, tool support becomes more feasible. In the current prototype, the deployment step is fully automated using the SmartFrog framework [16], enabled by the Grounded Model. Tools and techniques are being developed for the preceding stages. The current prototype employs a genetic algorithm for enumerating workload placement combinations and a Layered Queuing Model tool [3] for evaluation. In the future, use of a constraint solver [9] is planned for incorporating constraints in designs.

The Model Information Flow is an effort to introduce the same discipline into the definition, creation and management of enterprise IT systems as it has occurred in the business layer above where processes are defined, created and are managed largely based on models and designs. The expectation is that model-driven approaches in IT will also lead to shorter times needed for introducing new services and also for incorporating changes in IT faster and more accurately than today.

REFERENCES

[1] Scheer, A.W., Abolhassan, F., Jost, W., Kirchmer, M. (Eds.): *Business Process Automation*, ISBN 3540207945, Springer Verlag, 2004.

[2] IDS Scheer: *Aris*, http://www.ids-scheer.com.

[3] Rolia, J., Sevcik, K.C.: *The Method of Layers*, IEEE Trans. on Software Engineering, vol. 21, no. 8 pp. 689-700, August 1995.

[4] Graupner, S., Sahai, A.: *Policy-based Resource Topology Design*, 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK, May 9-12, 2005.

[5] Vose, M.D.: *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, Cambridge, MA, 1999.

[6] Curran, T.A., Keller, G., Ladd, A.: *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*, Enterprise Resource Planning Series, 320 pages, Prentice Hall PTR, July 28, 1997.

[7] *SAP Enterprise Service Architecture.*

[8] *SAP Business Blueprint Library for SAP Solution Manager*, http://help.sap.com/saphelp_sm40/helpdata/en/2a/62c33af63ae93ae10000000a11402f/content.htm.

[9] Ramshaw L, Sahai A, Saxe J, Singhal S.: *Cauldron: A Policy based Design Tool*, In the proceedings of IEEE Policy 2006.

[10] DMTF: *DMTF Virtualization System Profile* (work in progress), September 2006.

[11] *SAP Standard Application Benchmarks: Measuring in SAPS*, http://www.sap.com/solutions/benchmark/measuring/index.epx.

[12] Santos, C., Flores, P., Pruyne, J., Salazar, N., Zhu, X.: *Automated Application Component Placement at a Computing Utility*, INFORMS 2005 Annual Meeting, November, 2005.

[13] Santos, C.A., Sahai, A., Zhu, X., Beyer, D., Machiraju, V., Singhal, S.: *Policy-Based Resource Assignment in Utility Computing Environments*, DSOM 2004, Davis, CA, USA, November 15-17, 2004.

[14] Graupner, S., Andrzejak, A., Kotov, V., Trinks, H.: *Adaptive Service Placement Algorithms for Autonomous Service Networks*, in Brueckner, (Ed.): “Engineering Self-Organizing Systems”, LNCS 3464, Springer Verlag, May 2005.

[15] Rolia, J., Cherkasova, L., Arlitt, M., Andrzejak, A.: *A Capacity Management Service for Resource Pools*, WOSP 2005: 229-237.

[16] *SmartFrog*, http://www.smartfrog.org.