Managing software evolution is a complex task. Indeed, throughout their whole lifecycle, software systems are subject to changes to extend their functionalities, correct bugs, improve performance and quality, or adapt to their environment. If not evolved, software systems degrade, become obsolete or inadequate and be replaced. While unavoidable, software changes may engender several inconsistencies and system dysfunction if not analyzed and handled carefully hence leading to software degradation and phase-out.

This thesis proposes an approach to improve the evolution management activity in component-based software development processes. The solution adopts a Model-Driven Engineering (MDE) approach. It is based on Dedal, an Architecture Description Language (ADL) that explicitly separates software architecture descriptions into three abstraction levels: specification, configuration and assembly. These abstraction levels respectively correspond to the three major steps of component-based development (design, implementation and deployment) and trace architectural decisions all along development. Dedal hence efficiently supports evolution management: It enables to determine the level of change, analyze its impact and plan its execution in order to prevent architecture inconsistencies (erosion, drift, etc.).

Rigorous evolution management requires the formalization, on the one hand, of intra-level relations linking components within models corresponding to different architecture abstraction levels and on the other hand, of the formalization of inter-level relations linking models describing the same architecture at different abstraction levels. These relations enable the definition of the consistency and coherence properties that prove necessary for architecture correctness analysis. The evolution process therefore consists of three steps: First, change is initiated on an architecture description at a given abstraction level; then, the consistency of the impacted description is checked out and restored by triggering additional changes; finally, the global coherence of the architecture definitions is verified and restored by propagating changes to other abstraction levels.

Relations and properties are expressed in B, a set-theoretic and first-order logic language. They are applied on B formal ADL the meta-model of which is mapped to Dedal's and helps automatic model transformations. This integration enables to implement a development environment that combines the benefits of both MDE and formal approaches: Software architecture design using Dedal tools (graphical modeler) and architecture analysis and evolution management using B tools (animator, model-checker, solver).

In particular, we propose to use a B solver to automatically calculate evolution plans according to our approach. The solver explores a set of defined evolution rules that describe the change operations that can apply on architecture definitions. It automatically searches for a sequence of operations that both changes the architecture as requested and preserves architecture consistency and coherence properties. The feasibility of the evolution management approach is demonstrated through the experimentation of three evolution scenarios, each addressing a change at different abstraction level. The experimentation relies on an implementation of a search-based software engineering approach mixing software engineering and optimization and integrates our own solver with specific heuristics that significantly improve calculation time.