

Product Brochure

LS/TS – Living Systems® Technology Suite

Suite components

- ❑ **Run-time Environments:** several dependable run-time and integration environments, with administration tools, for J2SE and J2EE.
- ❑ **Development Tools:** covering all software engineering activities and phases of the development cycle, from agent-oriented analysis to deployment. Basis: Eclipse.
- ❑ **Development Methodology:** development process and modeling language applicable to different software agent paradigms. Basis: UML 2, Eclipse Process Framework (EPF).

Whitestein Technologies' Living Systems Technology Suite (LS/TS) is an industry-grade, Java-based foundation for the professional development and operation of products and solutions based on software agent technology and autonomic computing.

Builds Upon Your Infrastructure

LS/TS is compatible with your existing IT infrastructure. It seamlessly blends in and adds the features and functionality required to design, build, and operate robust and well-performing solutions that make use of the advanced properties of software agents, Multi Agent Systems (MAS), and autonomic computing. LS/TS is based on practice-proven object-oriented technologies, methodologies, and products.

This approach brings about the protection of your investments in infrastructure, know-how, and experience. And it ensures full compatibility with the offerings of major IT vendors, e.g., in the field of business process and data integration, as well as service-oriented architectures and messaging systems.

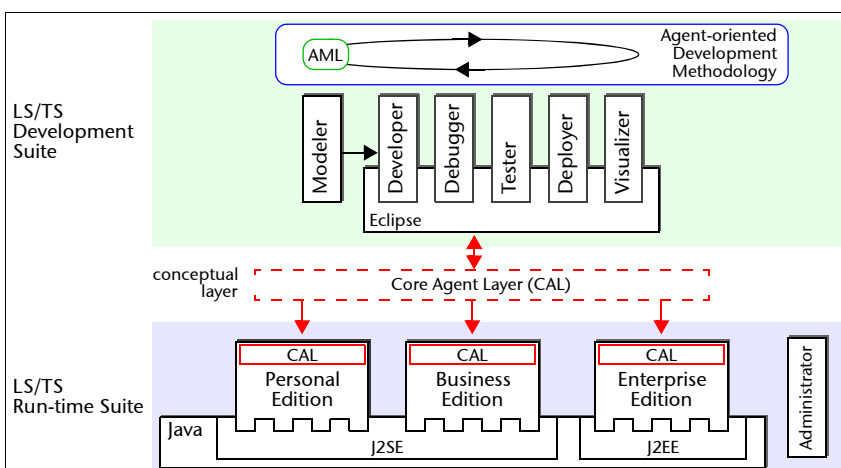
Grows with Your Needs

Different Editions are available for development and run-time purposes:

- ❑ **Personal Edition (PE):** for solution prototyping and exploration, development of small personal applications with basic run-time needs (run-time basis: J2SE).
- ❑ **Business Edition (BE):** for developing and operating highly demanding solutions and applications, with industry-level run-time features, such as resource control, transactions, or platform federation (run-time basis: J2SE).
- ❑ **Enterprise Edition (EE):** for developing and operating mission-critical systems, with the need for ultimate enterprise level functionality and run-time features, such as failsafe operation, clustering, integration via JCA and JMS (run-time basis: J2EE application servers).

Open and Future-proof

LS/TS was conceived with the know-how, experience, and practical needs of today's IT departments in mind. The tools build on Eclipse, the Run-time Environments extend the existing J2SE run-time environment and J2EE application servers.



To achieve clean, well-engineered system designs, and to allow for a flexible deployment in the suitable Run-time Environment, the Core Agent Layer (CAL) was designed. As a unifying layer, CAL defines the necessary abstractions and components to build agent systems, and is implemented in all LS/TS Run-time Environments to provide the core platform services.

The abstractions combine the expressiveness and power of the agent paradigm with the practical needs of software development and maintenance.

Figure 1: Architecture. The Living Systems Technology Suite provides a comprehensive, extensible Eclipse-based multi-user Development Suite to develop agent-based applications for the Java-based Run-time Environments of the Run-time Suite.

Product Suites – Offering Overview

Development vs. Run-time

To meet the needs of IT management and practice, the Editions of the Living Systems Technology Suite are available as Development and Run-time Suites (see also Table 1):

- ❑ **Development Suite:** for the development, integration, and deployment of agent-based software solutions and products. It comprises a set of development tools, the Agent-oriented Development Methodology (ADEM) and the Agent Modeling Language (AML), plus development versions of the corresponding Run-time Environments, incl. the Administrator tool.
- ❑ **Run-time Suite:** for the production operations and administration of solutions (BE and EE only). It comprises the corresponding Run-time Environments, plus the Administrator tool.

The components are described in more detail later in this document.

Component		Development Suite			Run-time Suite		
		PE	BE	EE	PE	BE	EE
Run-time environments & tools	Personal Run-time E.	✓	✓	✓	✓	✓	✓
	Business Run-time E.		✓	✓		✓	✓
	Enterprise Run-time E.			✓			✓
	Administrator	✓	✓	✓	✓	✓	✓
Development tools	Modeler		✓	✓			
	Developer	✓	✓	✓			
	Debugger		✓	✓			
	Tester		✓	✓			
	Deployer	✓	✓	✓			
Methodology	ADEM	✓	✓	✓			
	AML	✓	✓	✓			

Table 1: Product suites. The Business Edition (BE) and Enterprise Edition (EE) are available as Development Suite to engineer LS/TS based software solutions, and as Run-time Suite for the production use of these solutions. The Personal Edition (PE) is for exploration and prototyping only, and is not licensable for production use.

Support, Consulting, Training

The Business Edition and Enterprise Edition include start-up workshops for a smooth and successful start. Please contact us to discuss your specific support and training needs.

Licensing

The Suites are licensed as follows:

- ❑ **Development Suite:** per developer seat
- ❑ **Run-time Suite:** per CPU

Site or company licenses are also available. Please contact us for licensing details, as well as for specific offerings for use in non-profit research and exploration projects.

System Requirements

- ❑ The Developer, Debugger, Tester, Deployer, and Administrator tools of the **Development Suite** require J2SE 1.4 or newer, and are currently based on Eclipse 3.1. Eclipse is available for different platforms such as MS Windows, Linux, Solaris, AIX, HP-UX, and Mac OS X.
The current version of the Modeler is an add-in for Enterprise Architect, a third-party tool (www.sparxsystems.com.au), available for MS Windows and Linux platforms.
- ❑ The **Personal Run-time Environment** and the **Business Run-time Environment** require J2SE 1.4 or newer.
- ❑ The **Enterprise Run-time Environment** requires a J2EE 1.3 compliant application server, such as IBM WebSphere 5.x or 6.x, BEA WebLogic 8.x, JBoss 4.x.
- ❑ **Persistent data:** any JDBC-compliant database is supported, with more than 20 popular SQL dialects, including Oracle, DB2, Sybase, SQL Server, PostgreSQL, SAP DB (now MAX DB), and others.

Documentation

Product and developer documentation is available in English.

Run-time Suite

Three Editions

The LS/TS Run-time Suite provides different Run-time Environments – that is, middleware which adds the necessary enabling functionality to the base Java run-time environments needed to run agent-based autonomic systems and applications:

- ❑ Personal Edition (J2SE)
- ❑ Business Edition (J2SE)
- ❑ Enterprise Edition (J2EE)

Each of the environments provides the full functionality of the Core Agent Layer (CAL), however with differences in implementation sophistication.

This provides an easy upgrade or migration path to one of the other LS/TS Run-time Environments based on prevailing business needs.

The three environments differ regarding functionality in areas such as transaction management, client access, fail-safe operation, clustering, and resource management.

The Business Edition and Enterprise Edition can be combined for flexible deployment scenarios.

Please refer to Table 2 for a comparison of the main features.

Feature	Run-time Environment		
	Personal	Business	Enterprise
Full CAL support	✓	✓	✓
Data management with transactions		✓	✓
Non-persistent agent states	✓	✓	✓
Persistent agent states		✓	✓
Non-persistent messaging (agent-to-agent)	✓	✓	✓
Persistent messaging (agent-to-agent)			✓
Non-persistent Directory Facilitator (DF)	✓	✓	✓
Persistent Directory Facilitator (DF)		✓	✓
Client access through RMI	✓	✓	✓
Client access through Web services (SOAP)		✓	✓
Client access through JMS			✓
Resource management & control		✓	✓
Administrator tool	✓	✓	✓
Clustering support			✓
Federation support		✓	✓

Table 2: Features of the different LS/TS Run-time Environments. The Run-time Environments of the three Editions provide different sets of features

Administrator

The Run-time Suite is equipped with the Administrator, an Eclipse-based tool for the management and monitoring of the Run-time Environment and the agent applications running on top of it.

Flexible Deployment

The suitable production run-time environment – J2SE or J2EE – and the corresponding LS/TS Edition can be chosen at deployment time, as the applications developed with the Living Systems Development Suite are executable on both the Business and the Enterprise Run-time Environment.

Core Agent Layer (CAL)

The CAL is the unifying layer across all Run-time Environments, also serving as a clean interface to the development tools. The CAL defines and implements the agent abstractions and components needed by the software engineering practitioner to develop agent-based systems and applications that can be deployed using any of the three Run-time Editions.

The main CAL component is the Autonomous Agent, which can exhibit pro-active and goal-oriented behavior, and can flexibly communicate with the other agents to achieve coordination and cooperation. Other components include, among others, directory services, notification services, as well as access to the messaging infrastructure.

Semantic Communication

LS/TS' inter-agent message representation supports OWL and communicative acts such as requests, notifications, and queries, allowing complex domain models with multiple ontological views. With a simple, yet powerful API, the virtues of agent communication yield benefits across all aspects of LS/TS applications.

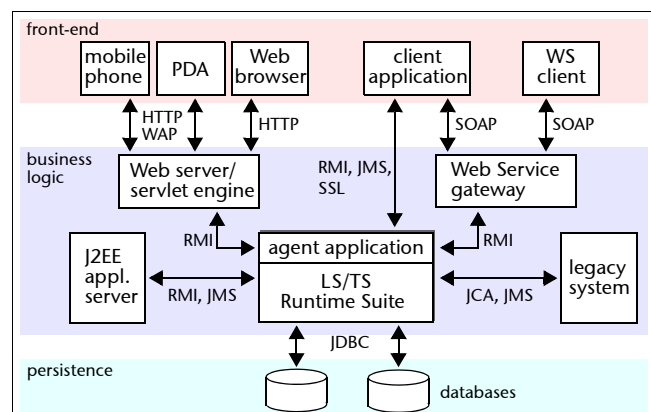


Figure 2: LS/TS n-tier solution architecture. The Java-based Run-time Environments enable the engineering, integration, and operation of state-of-the-art n-tier systems and solutions.

Execution Engines

LS/TS provides two execution engine types to implement the agent’s application logic. They can be extended and adapted, and also combined in a single application.

- ❑ **LISA:** the LISA execution engine has the task as its main concept, executing it using a continuation-based approach. A continuation states a possible next step to perform later, triggered by external events, logical conditions, or time. A LISA task can have sub-tasks (parallel and alternative branching). Agent interaction is supported with a messaging API and a system of interaction contexts that allows multiple concurrent conversations.
- ❑ **MARGE:** the MARGE execution engine enables the implementation of goal-oriented agents. Application developers provide agent capabilities and goals. The MARGE run-time selects and combines capabilities to obtain the desired results. MARGE uses the Belief-Desire-Intention (BDI) architecture. A MARGE agent keeps its knowledge in the belief base as logical propositions. The agent’s goals also have logical representations. Agent capabilities are plans, composed by a body, together with their relevance conditions (when a plan is applicable), and the achievement condition (what a plan will obtain). MARGE supports plan introspection through a process-algebraic API, allowing agents to reason about plans and tasks before executing them.

Security and User Management

The LS/TS Run-time Environment is able to enforce security policies, based on access control and sophisticated rights management for both agents and clients. The run-time provides full multi-user management with the Administrator tool.

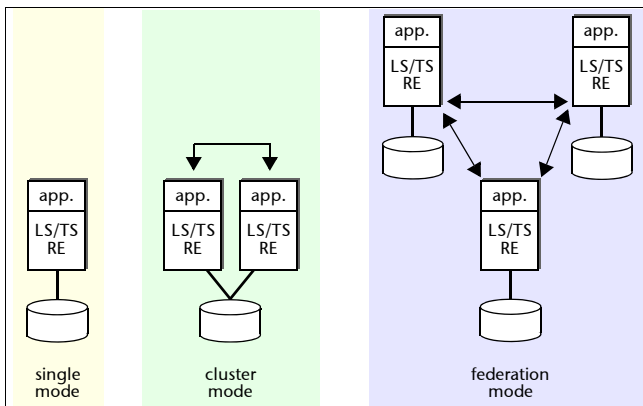


Figure 3: Operation modes. The LS/TS Run-time Environments (LS/TS RE) and applications (app.) can be operated in three different modes.

Operation Modes

For increased fault-tolerance, load-balancing or other architectural reasons, the Business and the Enterprise Run-time Environments can be run in different operation modes:

- ❑ **Single Mode:** the default set-up, where one single instance of the Run-time Environment executes the application.
- ❑ **Cluster Mode:** several Run-time Environments execute the same application in parallel, connected to the same databases. Should one node fail, the others take over transparently. Clustered nodes act as one logical system towards clients, other systems, and the application itself.
- ❑ **Federation Mode:** several Run-time Environments are loosely coupled and run separate parts of the application. Each node has its own databases, and operates as an independent entity towards external systems. The federation is self-configuring.

Resource Control

Fine-grained control of the available machine resources is of fundamental importance for the unobstructed, non-stop operations of systems and applications. The Business and the Enterprise Run-time Environment provide configurable resource pools and the corresponding monitoring functionality.

Client Access

The Living Systems Technology Suite provides various access technologies for clients (front-ends, other systems):

- ❑ **RMI:** all Editions provide an RMI stub to enable Web front-ends (via servlet engine), rich user clients, and other applications to connect and communicate using messages. The external system or front-end appears as an agent.
- ❑ **Web Services (SOAP):** the Business and Enterprise Editions are equipped with a Web services gateway, allowing existing systems or rich user clients to connect via SOAP communication.
- ❑ **JMS:** the Enterprise Edition is equipped with system-level JMS support for interoperability with other systems and clients. The Business Edition supports JMS connectivity on the application level.

Development Suite

Overview

The development tools are easily installable and upgradable plug-ins to the Eclipse platform (www.eclipse.org), with the exception of the Modeler, which is an add-in for Enterprise Architect (www.sparxsystems.com.au).

Developer

The Developer is the main design and implementation tool for the application programmer. It is an extension to the standard Eclipse Java programming environment, providing specific navigators, views, and manipulators of the Java code for the CAL and the various types of agent execution engines.

- ❑ **Explorers:** allow easy navigation through the various agent abstractions and CAL components.
- ❑ **Creation wizards:** help to quickly and easily create new projects or develop new CAL components.
- ❑ **Extended search:** systematically find specific CAL components.
- ❑ **Code templates:** allow fast development of frequently used code structures.

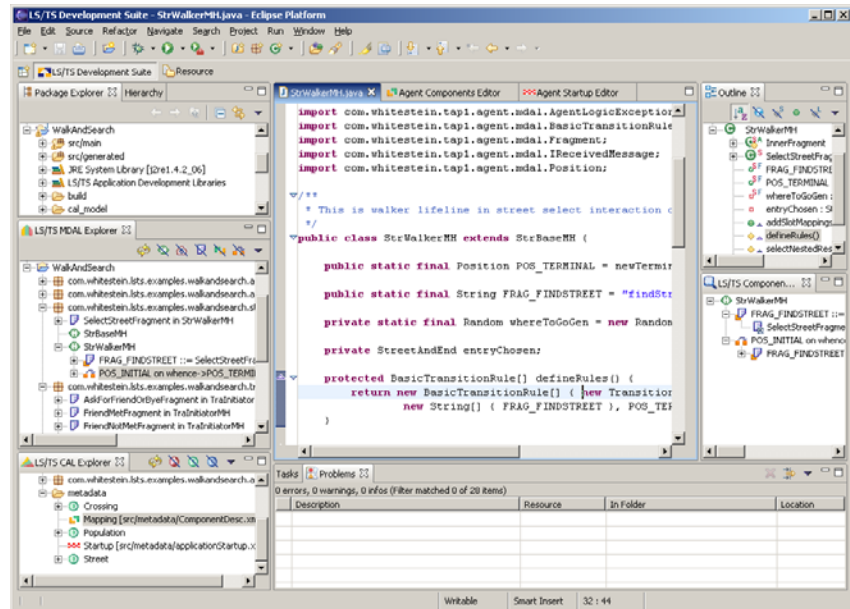


Figure 4: Developer screen. The Developer provides different views and specialized tools for creating, structuring, programming, and inspecting agent-based applications and components.

Debugger

The Debugger serves the developer in locating and repairing defects in the application code. It is an extension to the standard Eclipse Java debugger, with direct support for the agent abstractions of the CAL and the agent's execution engines. Local and remote debugging is supported.

- ❑ **Extended breakpoints:** in addition to conventional breakpoints set on a specific line of code, breakpoints can also be triggered by events within the system, e.g., upon sending or receiving a specific message, on read or write access to a specific variable of the agent's state, and more.
- ❑ **Target instances, not only classes:** further extending conventional breakpoints, all breakpoints can be set for a specific instance of an agent type, not just the agent type (agent class) itself.
- ❑ **Flexible platform control:** when the breakpoint is triggered, only the threads of the agent application are paused, allowing to inspect the message queues and the agent states with the Administrator tool.
- ❑ **Message history:** in combination with the Administrator tool, the history of already handled messages can be browsed and inspected.

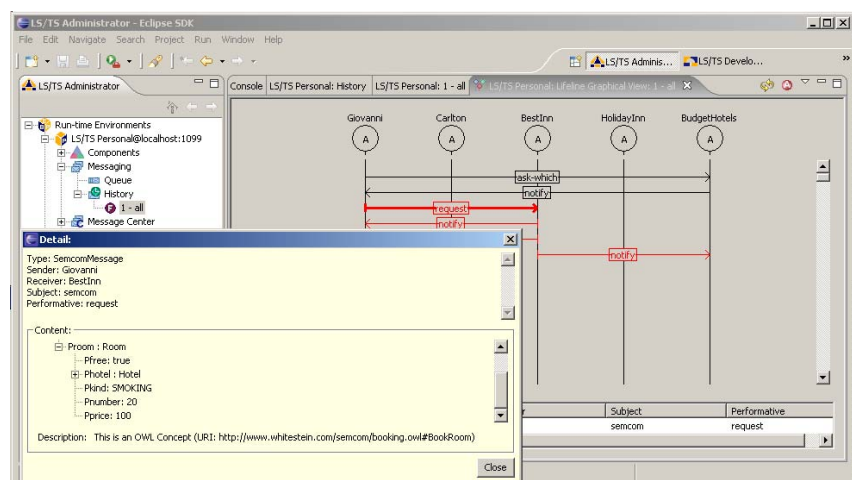


Figure 5: Lifeline view. The Administrator features on-line tracking of the messages exchanged among LS/TS agents. Several filters and views are available, such as the lifeline and detail view.

Deployer

The Deployer assists the developer in configuring the completed agent application for deployment to one of the LS/TS Run-time Environments.

- ❑ **Configuration of Run-time Environment properties:** e.g., the persistence layer (e.g., Hibernate, database).
- ❑ **Configuration of agent application properties:** such as persistence of agent and message states, initial startup values.
- ❑ **Automatic packaging:** with configured libraries and configuration files.

Administrator

Besides its functionality to configure, monitor and otherwise operate the LS/TS Run-time Environment (see Run-time Suite), the Administrator also offers useful functionality for finding errors, analyzing problems, and debugging, such as the lifeline and detail view (see Figure 5).

Tester

The Tester framework supports unit testing of agent components, single agents, and their integration with the environment.

- ❑ **JUnit:** the Tester framework is based on the JUnit framework.
- ❑ **Testing interactions:** support for testing the roles in complex interactions between agents.
- ❑ **Mock support** for Servants and Data Access Objects.

Modeler

The Modeler serves the analyst and designer in building AML models of the agent application.

- ❑ **AML modeling:** enables the creation of platform independent models of multi-agent systems by means of AML and UML modeling constructs and diagrams.
- ❑ **Modeling of LS/TS applications:** through additional LS/TS-specific modeling constructs and application of specific modeling guidelines.
- ❑ **Inherits features from Enterprise Architect:** such as comprehensive support for UML 2, flexible documentation generation, forward and reverse code engineering, support for testing, support for model maintenance, import/export, multi-user capabilities, etc.

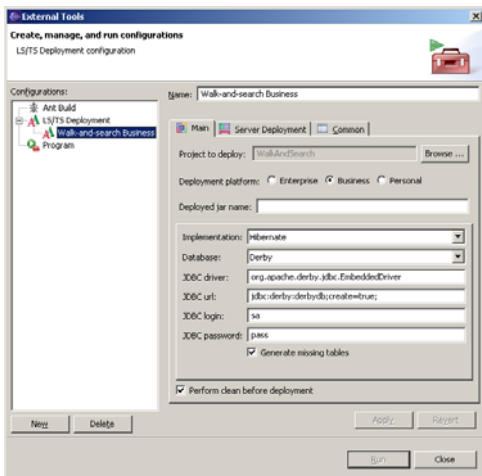


Figure 6: Deployer screen. The Deployer tool supports the software engineer in configuring and packaging an application for a specific target LS/TS Run-time Environment.

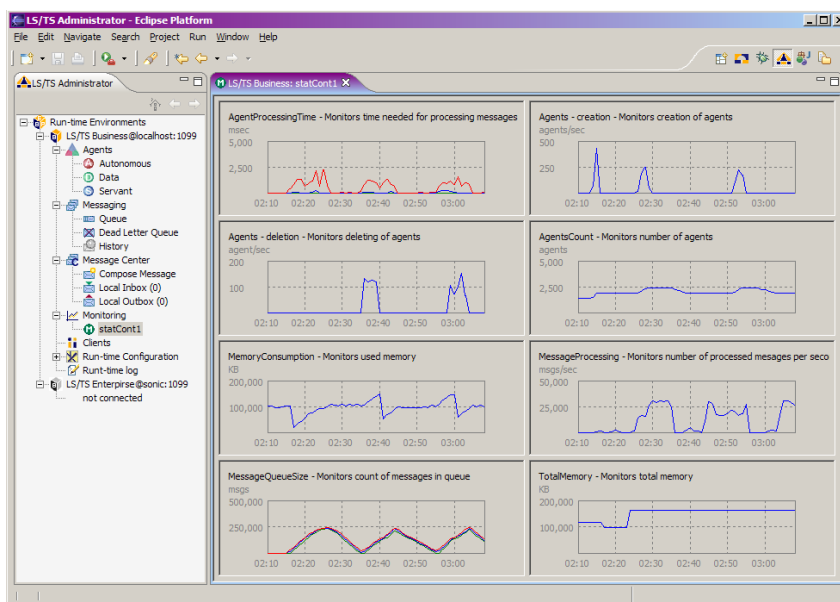


Figure 7: Administrator screen. The Administrator tool serves to configure, monitor, and otherwise operate the Run-time Environment, as well as for analyzing problems and for debugging.

Methodology (ADEM, AML)

The Methodology comprises two main components: ADEM, the Agent-oriented Development Methodology, and AML, the Agent Modeling Language.

ADEM is a comprehensive agent-oriented methodology for the specification, development, deployment, and operation of LS/TS systems and applications (see Figure 8).

ADEM supports Situational Method Engineering, and can easily be integrated with existing development processes, such as Rational Unified Process, Agile Unified Process, Enterprise Unified Process, OPEN Process Framework, and others.

Furthermore, ADEM implements an iterative, incremental approach to system development via clearly defined milestones.

AML is an agent-specific extension to UML 2. AML is designed to support business modeling, requirements specification, analysis, and design of software systems based on software agent concepts and principles.

AML supports the abstraction of the structural and behavioral concepts of Multi Agent Systems (MAS), such as:

- ❑ ontologies;
- ❑ MAS entities (agents, resources, environments);
- ❑ social aspects;
- ❑ behavior abstraction and decomposition;
- ❑ communicative interactions;
- ❑ services;
- ❑ observations and effecting interactions;
- ❑ mental aspects used for modeling mental attitudes of entities;
- ❑ MAS deployment;
- ❑ agent mobility.

To cope with the practical needs and existing infrastructures of IT departments, AML can be implemented with both UML 1.x and 2.x based CASE tools.

Furthermore, emphasis was put on the extensibility of AML: using UML profiles, users can define own language extensions to customize AML for specific modeling approaches and techniques, software agent theories and technologies, and particular domains.

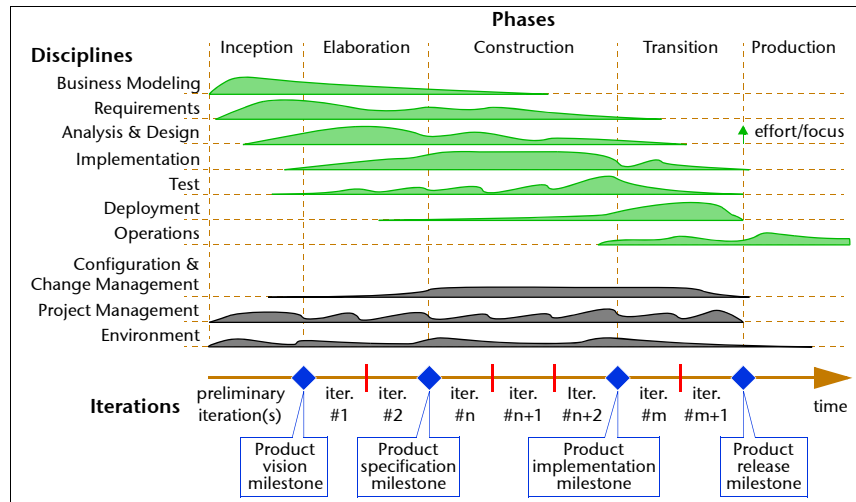


Figure 8: Agent-oriented Development Methodology (ADEM), a comprehensive agent-oriented methodology for the specification, implementation, deployment, and operation of LS/TS-based systems and applications. The figure indicates the efforts invested and thus the activities focus in each stage of a project.

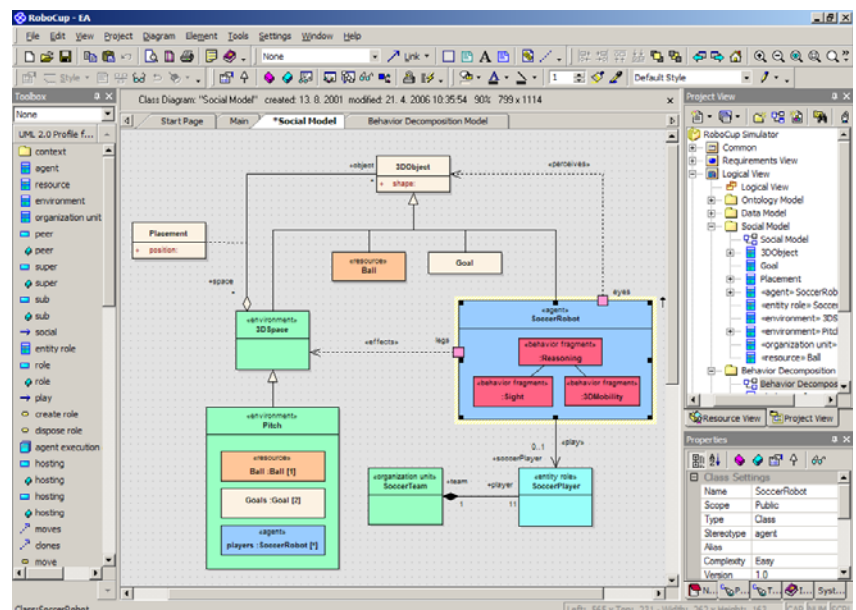


Figure 9: Modeler screen with AML Model. Analysts use the Modeler to conceptually model the multi agent system and its parts using AML, the UML 2 based Agent Modeling Language.

Contact Information

Whitestein Technologies AG
Tödistrasse 23
8002 Zürich
Switzerland
Phone +41 44-256-5000
Fax +41 44-256-5001
E-mail info@whitestein.com

For more information, please also visit our Web site at

→ <http://www.whitestein.com>.

About *Whitestein Technologies*

Whitestein Technologies is a leading innovator in the area of software agent technologies and autonomic computing & communications. *Whitestein Technologies'* product offering includes advanced solutions for the telecom, logistics, financial services, and business process management domains, as well as a comprehensive middleware for the development and operation of autonomous systems. *Whitestein Technologies'* customers and partners include leading global enterprises in the above markets, as well as technology companies, system integrators, universities, and other research institutions.

Whitestein Technologies was founded in 1999 and is privately held. The firm is headquartered in Zug (Switzerland) with offices in Zürich (Switzerland), Donaueschingen (Germany), and Bratislava (Slovakia).

All information herein are subject to change without further notice.

Whitestein Technologies, Living Systems, and the corresponding logos are registered trademarks of Whitestein Information Technology Group AG.

All company, product, or service names may be trademarks or service marks of their respective holders.