Project no.	034084
Project acronym:	SELFMAN
Project title:	*Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components*

European Sixth Framework Programme
Priority 2, Information Society Technologies

Deliverable reference number and title:	D.5.1 User Requirements
Due date of deliverable:	v0 July 15, 2007 - v1 November 30, 2007
Actual submission date:	v0 July 15, 2007 - v1 November 30, 2007
Start date of project:	June 1, 2006
Duration:	36 months
Organisation name of lead contractor for this deliverable:	FT
Revision:	1.2
Dissemination level:	CO

Contents

1 Executive summary **1**

2 Contractors contributing to the Deliverable **3**

3 Results **5**
3.1 Proposed templates for user requirements 5
3.1.1 Applicative Contexts 5
3.1.2 Use Cases 5
3.1.3 Requirements 6
3.2 Multi Service M2M Application 8
3.2.1 Applicative Context 8
3.2.2 Use Cases 23
3.2.3 Requirements 49
3.2.4 Conclusion 70
3.3 Distributed Database Use Case 72
3.3.1 Applicative Context 72
3.3.2 Scenarios 72
3.3.3 Autonomic Scenarios 80
3.3.4 Requirements 81
3.3.5 Conclusion 82
3.4 P2P TV Application 83
3.4.1 Applicative Context 83
3.4.2 Use Cases 83
3.4.3 Requirements 83
3.4.4 Conclusion 84

4 Conclusion on T5.1/D5.1 **85**

5 Appendices: publications and other documents produced by T5.1 **87**

1 Executive summary

The purpose of the WP5 is:

1. to provide use cases and requirements in different applicative contexts (applications),
2. so to help the Selfman project as a whole to choose which application(s) will be demonstrated,
3. and then to perform evaluations of the Selfman demonstrators and more gloablly the Selfman technologies and overall approach (autonomics based on components and overlays).

User requirements (D5.1) for each applications actually cover the description of an applicative context (application), the proposition of some (autonomic) use cases (scenarios) in this context and finally functional/non functional and operational requirements on components, transactions, overlay networks and self-* features (resp. WP1 to WP4) associated to the implementation of the proposed use cases in the considered applications.

Four applications were considered during Task 5.1[1]. The first one proposed by France Telecom concerns M2M systems. The second one proposed by ZIB concerns a distributed database system. Two additional applications were investigated as replacements for the one that should have been proposed for the partner E-plus which left the Selfman project in its first year. The first one, proposed by the Staak company (previously named PeerTV) (contact established by KTH(P2)), concerns P2P video streaming (P2P TV). The second one, proposed by the Bull company (contact established by France Telecom R&D(P4)), concerns a J2EE application server. After investigation, the latter applications from Bull finally appeared not suitable for the Selfman project and was then discarded (more details in the conclusion of this deliverable). The former application by Staak is still under investigation. Staak should join the Selfman consortium.

The M2M application was developed in collaboration with WP2 (components) and in connection with WP3 (transactions) and WP1 (overlay networks, security). Its goal is to develop large scale distributed M2M systems. The illustrative M2M system, that is basically dedicated to the management of the thermal environment of buildings is responsible for transporting data from very numerous sensors to several M2M services which process data from sensors and send commands to the thermal equipments actuators. The M2M use case exhibits strong requirements towards autonomic (self-*) features and components ; and secondary requirements towards transactions and overlay networks.

The distributed database application was developed in collaboration with WP3. Its goal is to develop a distributed transactional database which supports versioning. A wiki , which is the illustrative example, can then be added by a thin layer on top of it. The wiki distributed database application exhibits strong requirements towards autonomic (self-*) features, transactions and overlay networks ; and secondary requirements towards components.

[1] More elements on the history/process of Selfman WP5 in the 1st year of the project is detailed in Section 4.

The P2PTV application was developed in collaboration with WP1 and especially KTH. Its goal is to develop a P2P system for streaming of videos and TV. The P2PTV exhibits strong requirements towards overlay networks and self-* properties, secondary requirements on components and no requirements on transactions.

2 Contractors contributing to the Deliverable

France Telecom R&D(P4), ZIB(P5), KTH(P2)and PeerTV have contributed to this deliverable.

France Telecom R&D(P4) France Telecom has contributed on the definition of the M2M application, use cases and associated requirements and has investigated the J2EE application with the Bull company[2]. This work is made in connection with the work in WP2 on the Selfman architectural framework in which France Telecom contributes by several extensions of the Fractal component model (probes, composite probes, event-condition-action decision making rules, transactional components reconfiguration, large scale management architecture), WP1 (overlays, security) and WP3 (transactions on overlays).

France Telecom is leader of the WP5, editor of the Selfman wiki section devoted to WP5 and editor of this deliverable (T. Coupaye). France Telecom contributors to this deliverable are (in alphabetical order): O. Beyler (R&D Engineer, M2M platforms), B. Dillenseger (Senior Researcher, performance evaluation), T. Coupaye (Senior Researcher, architecture), A. Diaconescu (Junior Researcher, composite probes), A. Harbaoui (PhD Student, autonomic testing), N. Jayaprakash (PhD Student, decision making)), M. Kessis (PhD Student, large scale management), M. Lacoste (Researcher, security), A. Lefebvre (Senior Researcher, large scale management), M. Leger (PhD Student, transactional reconfiguration), F.-G. Ottogalli (R&D Engineer, M2M platforms architecture).

There have been no deviations from the workplan but France Telecom consumed more human resources than expected on this task/deliverable. Indeed, the work on user requirements for M2M application actually revealed itself more complex than forecasted. France Telecom does develop and exploit operationally M2M platforms and applications but it is worth mentioning that today's operational M2M generic platforms are in fact pretty basic with typically a few dozens to hundreds sensors sending data directly to one client service/application that consumes/processes this data. The M2M use case proposed in this Selfman deliverable explicitly tries to envision tomorrow's M2M systems that are expected to be of much higher size and complexity together with a concept of multi services M2M platform where data may be share by several M2M client services - and that then would required advanced features such as component-based architectures, overlay networks, transactions and autonomic (self-*) properties studied in Selfman. Such reflections required a greater effort than expected.

ZIB(P5) has contributed on the definition of the wiki application, use cases and associated requirements. This work is based on the development in WP3 on transactions in structured overlay networks.

ZIB contributors to this deliverable are (in alphabetical order): M. Moser (PhD Student), S. Plantikow (PhD Student), T. Schütt (PhD Student).

There have been no deviations from the workplan, but due to E-Plus leaving, their input to the distributed database scenario is limited. Especially, in the areas

[2] which tales a significant amount of time and ressources that do not appear in the deliverable...

of performance requirements and user profiles, real-world data would have been helpful.

Staak (PeerTV) together with KTH(P2) has contributed on the definition of the P2PTV application.

Staak contributors are: Andreas Dahlström, Johan Ljunberg, Sameh El-Ansary, Mohammed El-Beltagy together with Seif Haridi from KTH.

There have been no deviations from the workplan since PeerTV and the P2PTV application was not included in the initial workplan. This use still being under consideration in the Selfman project as a replacement for the E-Plus leaving.

3 Results

This section specifies user requirements for each of the 3 applications: Multi Service M2M, Wiki Distributed Database and P2P TV.

More precisely, each application covers:

- the description of an applicative context (or application)
- the description of some (autonomic) use cases (scenarios) in this context
- and associated functional/non functional and operational requirements associated to the implementation of the proposed use cases in the considered applicative contexts. Operational requirements can be taken as technical requirements on components, transactions, overlay networks and self-* features, typically provided by Selfman working packages 1 to 4.

Important notice: The use cases and requirements given in this deliverable are extensive. They are not to take as commitments by the Selfman project. Use cases and associated requirements that will be supported will be defined in the sequel of the project.

3.1 Proposed templates for user requirements

This section describes the templates proposed to describe applicative contexts, use cases and requirements for the 3 considered applications. These templates come in the form of ID Cards. They are based on common forms of use cases and requirements with minor adaptations to the Selfman context e.g. the 'scope' field in use cases and the 'target' field in requirements.

3.1.1 Applicative Contexts

The descriptions of applicative contexts are given in free text with possible use of UML context/actor and activity diagrams.

3.1.2 Use Cases

Use cases are more formally expressed with the following template or *ID Card*.

(Mis)suse Case: The name of the (mis)use case. Misusecases are use cases from an hostile actor point of view. Use case names are built with the following lexical conventions: {UC}.{MSM2M, WikiDB, P2PTV}.name

where MSM2M refers to the Multi Service M2M application, WikiDB the wiki distributed database application and P2PTV the P2P video streaming application.

Scope: One or several (by order of importance) among M2MServices, WikiService, P2PTVService, M2MMiddleware, DDBMiddleware, P2PTVMiddleware, Selfman middleware where M2MServices, WikiService and P2PTVService refer to the

applicative or service layer of the 3 applications, M2MMiddleware, DDBMiddleware and P2PMiddleware refer the specific middleware underlying each application: M2M Middleware, Distributed DataBase (DDB) Middleware for ZIB, P2P Middleware for Staak ; and Selfman middleware refers to middleware produced by Selfman typically P2P overlay, transactions, components, self-* services (support for repair, optimization, etc.). NB: this is also a way of showing links with other Selfman WPs. Selfman middleware can be precised as selfman/components, selfman/overlays, selfman/self-* and selfman transactions if needed.

Description: Free text description possibly with schemas.

Primary actor: One among the actors defined in each applicative context e.g. end user, architect, programmer, deployer, administrator.

Stakeholders: A set of actors (can be complemented by UML actor diagrams). Can be empty.

Preconditions: A condition that must be satisfied to fire start the use case

Trigger: The event that causes the use case to be initiated. This event can be external, temporal or internal.

Basic course of event: A list of steps that describe the happy scenario . Can be written as an bullet list and/or UML sequence diagram.

Alternative path: Use cases may contain secondary paths or alternative scenarios, which are variations on the main theme.

Postcondition: A condition that is ensured if the use case sucessfully ends.

Miscellaneous: All other relevant pieces of information that do not fit in other fields. This section if very often empty.

3.1.3 Requirements

Requirement: The name (unique ID) of the requirement. Requirement names are built with the following lexical conventions: RQ.MultiServicesM2M, WikiDB, P2PTV.*name*

Use case(s): The use case(s) from which the requirement has been derived. This field cannot be empty for requirements always come from use cases.

Priority: One among MAY, SHOULD, MUST. MAY is means 'optional': "it would be nice if the system would have this feature but there is no harm if it does not". SHOULD means 'recommanded': "there may exist valid reasons for the system not to support this feature but the full implications must be clearly understood and carefully weighed before choosong a diffreent course". MUST means 'required': "this feature is an absolutely required in the system".

Description: Free text description (basically what is in the current version of the deliverable).

Rationale: The reason why the requirement is necessary.

Type functional and data, performance, operational, security + "look, feel & use", business, legal, standards if needed Functional and data represent applicative requirements. Operational refers to requirements on the expected technological environment -¿ typically requirements on other WPs components, overlays, transactions, self-reprair, self-protect, self-optimize, self-configure NB: In the functional and performance requirements, we find what was tagged "properties" in the current deliverable. In operational requirements, we find what was tagged "mechanims" in the current deliverable. This should made both D5.1 contributors and evaluators happy! When used, 'operational' should be sub-classed as operational/overlay, operational/components, operational/transactions, operational/self-*

Dependency: A reference to one or several other requirements with possible short explanations. Often empty.

Assessment: Criterias, processes that will be used to validate or invalidate the support of the requirement.

Target: The Selfman WP and if possible more precise Selfman work concerned.

Miscellaneous: All other relevant pieces of information that do not fit in other fields

3.2 Multi Service M2M Application

This section introduces the Machine-To-Machine (M2M) use case proposed by France Telecom. The use case is made of an applicative context, some autonomic scenarios in this context and finally the requirements in terms of i) functional and non functional desired properties of a M2M system and ii) the required mechanisms to support the proposed scenarios. The latter part is to be taken as requirement to the other Selfman Working Packages and especially as requirement on components (WP2), autonomics features (WP4), transactions (WP2) and overlay networks (WP1).

3.2.1 Applicative Context

The applicative context we consider is that of a large M2M (Machine-To-Machine) system dedicated (essentially but not only) to the management of the thermal environment of buildings (homes). The big picture of the considered M2M system is given in Figure 1.

The M2M system is responsible for transporting data from sensors to several M2M services which in turn process data and send commands to the thermal equipments. At the edges of the system, thousands (or millions) of buidlings are equiped with sensors such as thermometers and smoke detectors ; and actuators on thermal equipments such as heaters and boilers. All these data are presents in a privat e local area network (called the *domestic environment* in the sequel). A gateway is present in this environment which role is to export these data to interested M2M services, provided by third party service providers, that will use process/use these data. Gateways ('GW' on Figure 1) can be seen as peripheral nodes of the M2M system. Three M2M services are considered in the use case. A fire alarm service detects fires by means of in-houses smoke detectors and possibly correlations with the thermal regulation service. A weather forecast service provides local, regional and national data about current (observed) and forecast weather. A thermal regulation service uses data from in-houses sensors (e.g. thermometers) and possibly weather forecasts to provide a thermal regulation of the domestic environments by sending commands to in-houses thermal equipments (actuators on boilers, radiators, etc.).

Different actors appear in the M2M use case:

- individuals (end-users) who want their domestic thermal environment to be managed (including fire detection). They of course agree to have their homes equipped with sensors (e.g. thermometers, smoke detectors) and actuators on their thermal equipments (e.g. boilers, radiators). They also agree to have their sensors data exported into the M2M infrastructure so as to be possibly used by third party service providers.
- M2M service providers: thermal regulation, fire alarm and weather forecast.
- the infrastructure operator who is in charge of operating the system, i.e. of deploying and managing the infrastructure so as to guarantee its correct behaviour (including QoS).

The M2M system architecture is typically a data-flow (Pipe & Filter) architecture made of interconnected nodes that receive, process and send data. The

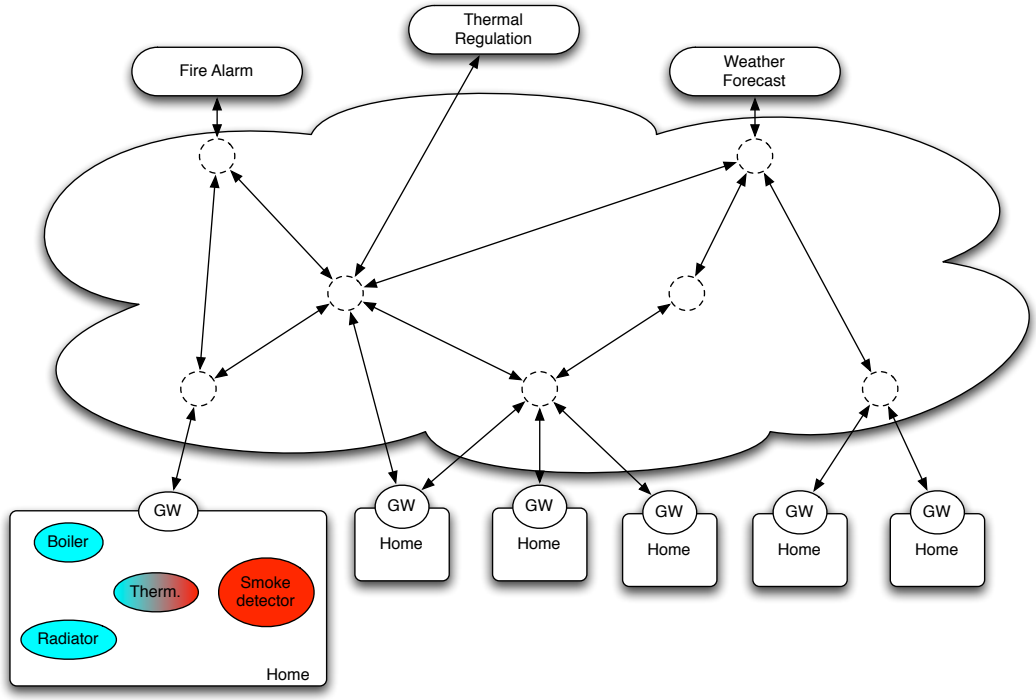


Figure 1: Big picture of a multi serice M2M application.

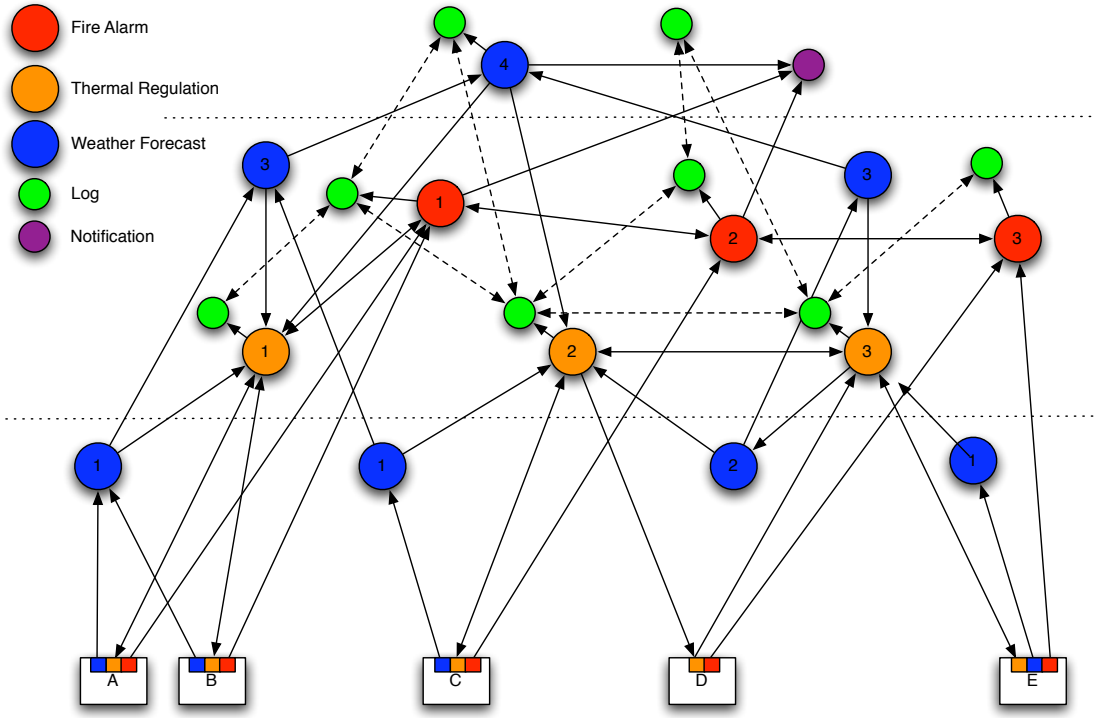


Figure 2: Architectural view of a archetypal multi service M2M application.

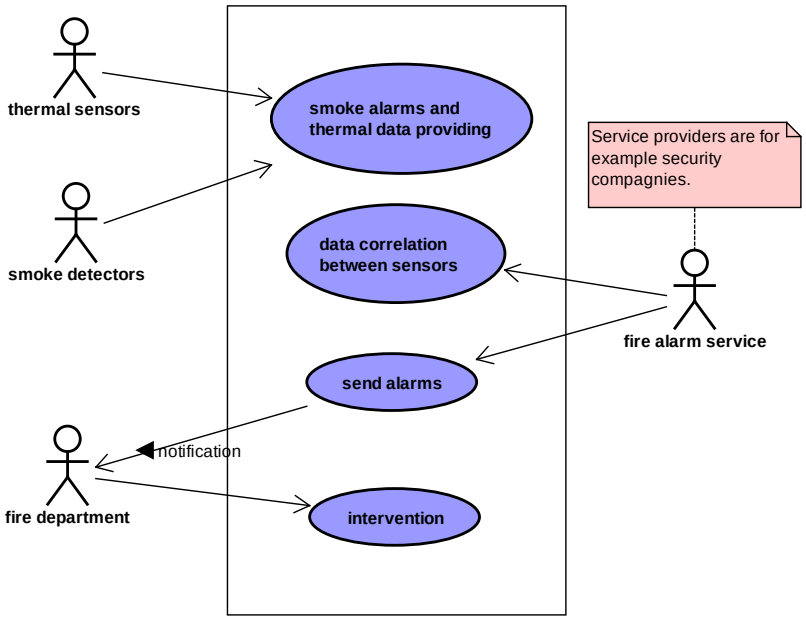


Figure 3: Fire Alarm service context diagram.

complete system (see Figure 2) involves 5 interacting services: 3 are applicative M2M services: *thermal regulation*, *fire alarm*, *weather forescast*, and 2 infrastructure (technical, generic) services (or *enablers*): *logging* and *notification*. The services are distributed: each service is implemented of a set of interconnected distributed nodes. The complete M2M system is made of the interconnected service nodes.

An important feature is that different criticities are associated to services (and hence possibly to the data they manipulate): the fire alarm service is of higher priority than the thermal regulation service which is itself of higher priority than weather forecast service. Also, data emitted from the domestic environment could be categorized as data devoted to a unique service versus data shared among services. For instance, temperature data are used by thermal regulation, fire alarm and weather forecast ; while smoke detection data is used only by fire alarm. Due to the different criticities and business decisions between service providers to share or not data, M2M infrastructure elements (service nodes) may be shared or not between services. Another feature, due to the large scale of the system, is the hierarchical data diffusion pattern used - with typically 3 layers: local, regional, national (separated by dotted lines in Figure 2).

Fire alarm The purpose of the fire alarm service is to detect fire situations in domestic environments and to notify these alarms to third parties, typically the appropriate fire departments.

Fire detection is based either on smoke detectors only (by default), or by correlating data from smoke detectors and temperature data from thermal regulation. Indeed, to prevent false alarm detection, thermal regulation can help in discriminating situations (see Figure 6). Smoke can be detected in a house because of overcooked food in the oven (!), but without significant temperature increase. This

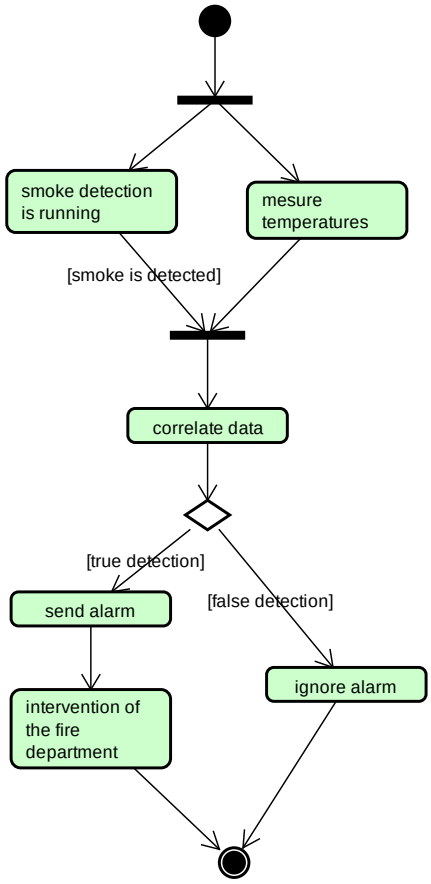


Figure 4: Fire Alarm service activity diagram.

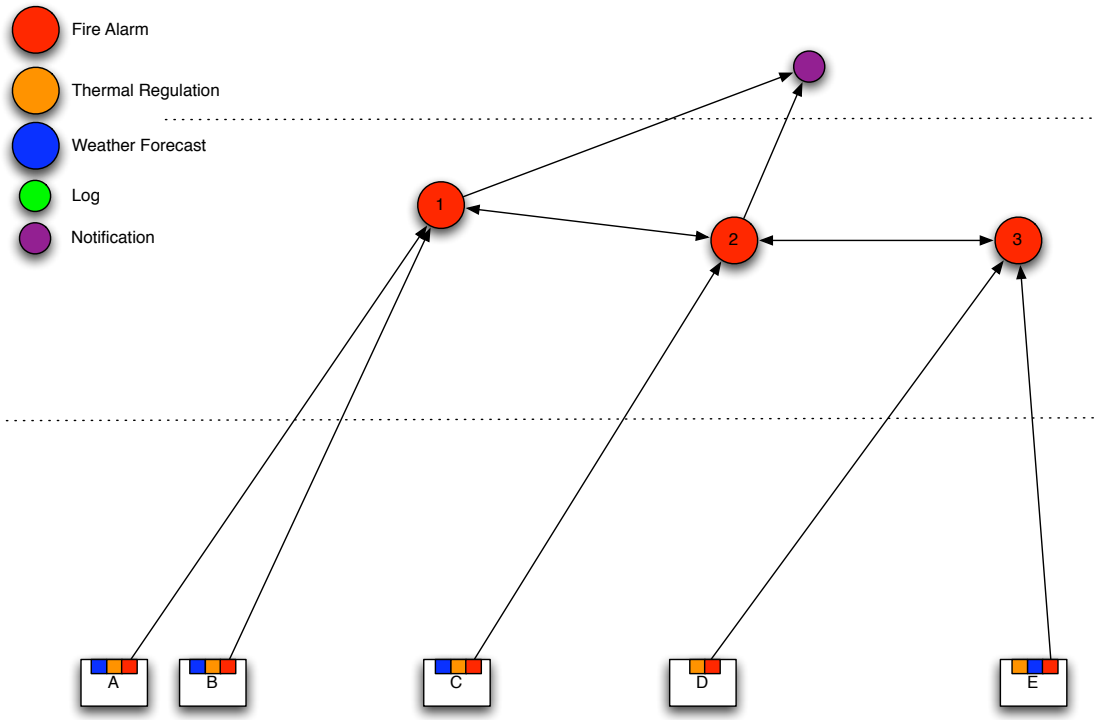


Figure 5: Fire Alarm service architecture.

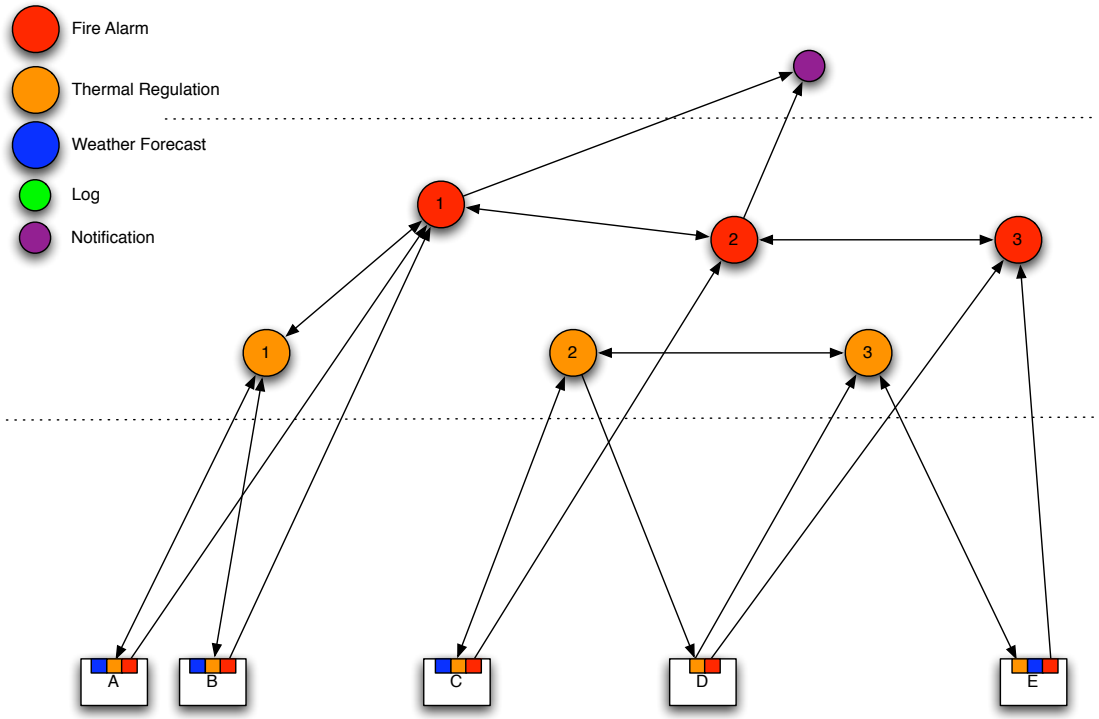


Figure 6: Fire alarm use case.

situation could probably be classified as a false fire alarm detection. In the other hand, if the thermal regulation service provides in temperature historical representing a sharp increase, this probably describe a true fire alarm detection even without smoke detection. In a basic way, fire detection is based on smoke detectors. When smoke is detected in a house, a fire alarm node (red node in Figures 5 and 6) receives an alarm message from it and then a notification is sent. This situation is depicted by red node 2 in Figure 5. Node 3 is not able to notify by it self. It has to delegate the notification scenario to an alarm node linked to the notification service (nodes 1 or 2 in this example).

As the fire alarm service is critical and prioritary, data processed by fire alarm service nodes are of high priority. Moreover, in overload situations of the fire alarm service, resources can be 'stolen' from other services. In order to (try to) prevent such situations, self-optimization mechanisms such as intra-service routing or load-balancing can take place. Intra-service routing depicts situations in which an overloaded node would send the data it receives to a pair node belonging to the same service, e.g. the fire alarm service in this case. Finally, a specific network of nodes is dedicated to the fire alarm service (cf. Figure 5) with specific resources dedicated to it. A constraint on the fire alarm service which is critical us that the sub-graph of the complete system graph corresponding to the fire alarm shoud always be connected.

Weather forecast The weather forecast service provides data about the current weather and future tendencies. The forecasts are build upon actually observed data provided by local sensors (thermometers, barometers) and possibly satellical data (cf. Figure 9). Local data are more accurate but with a narrow range of validity. Forecasts based on it are for short periods of time (typically couple of hours). Regional data aggregate local data. It help to estimate longer period forecasts (typically about one day). The national level uses both regional data and satellites data. Forecasts can be made on longer periods such as several days.

The weather service is organized hierarchically to match the accuracy of the weather forecasts discussed above. In each level (local, regional, national), nodes are functionally equivalents, i.e. they implement the same function/algorithm

A strong connectivity hypothesis exists between the regional and national levels. No hypothesis exist between local and regional levels. A node can be isolated and so need nodes from other services to keep the connectivity. This is QoS and pricing concerns.

Weather forecast can be used in several ways. The most simple one is to provide weather forecasts to customers (subscribers). In that case, a weather forecast node have to notify the forecast to a customer thanks to a notification service (e.g. blue node 4 in Figure 10). Another way is to provide weather forecast to the thermal regulation service: thermal regulation is essentially based on the actual observed local temperatures, but it can include weather forecasts so as to anticipate evolutions of the weather. The accuracy and the time validity of the forecasts depend on which node is providing the data. On the Figure 10, nodes 1 and 2 provide local data with a short range of validity. Nodes 3 give accurate forecasts about a region with a middle range of validity. Node 4 gives national level forecast with a longer range.

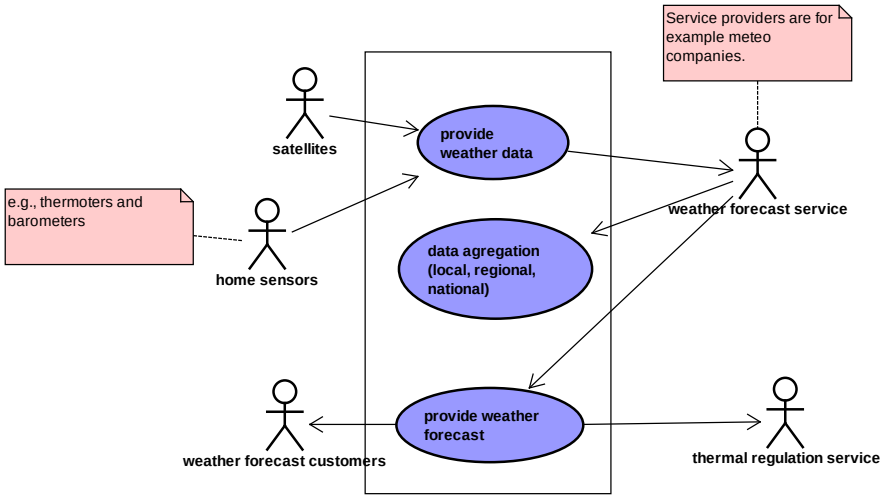


Figure 7: Weather forecast service context diagram.

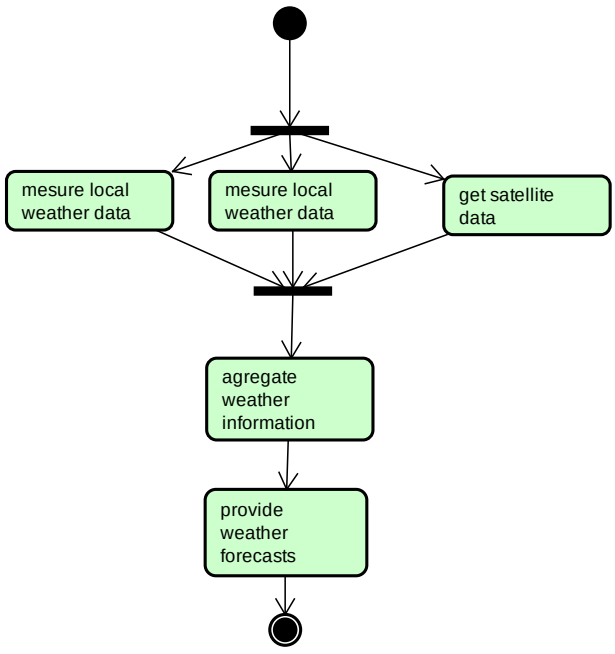


Figure 8: Weather forecast service activity diagram.

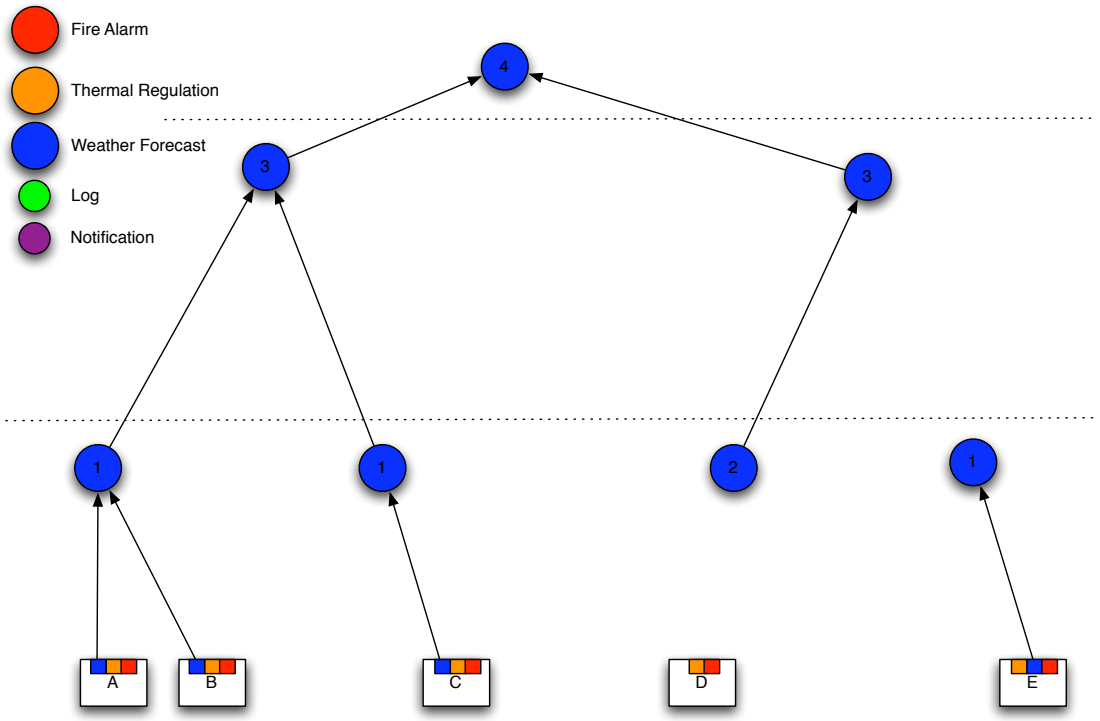


Figure 9: Weather forecast architecture.

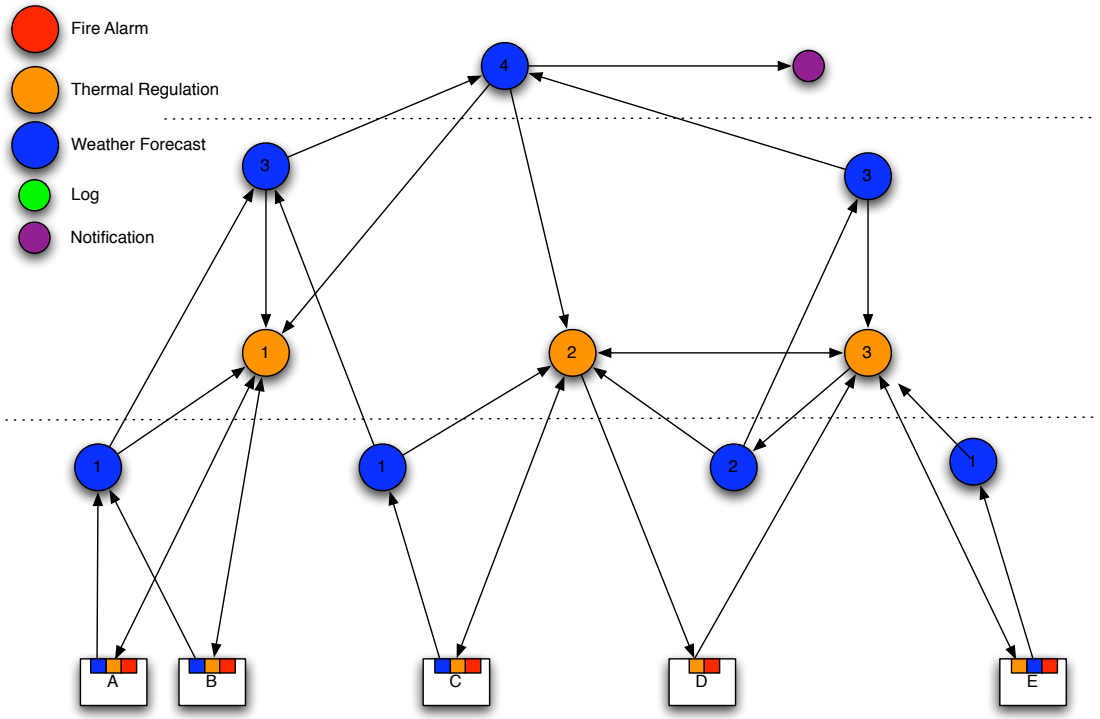


Figure 10: Weather forecast use case.

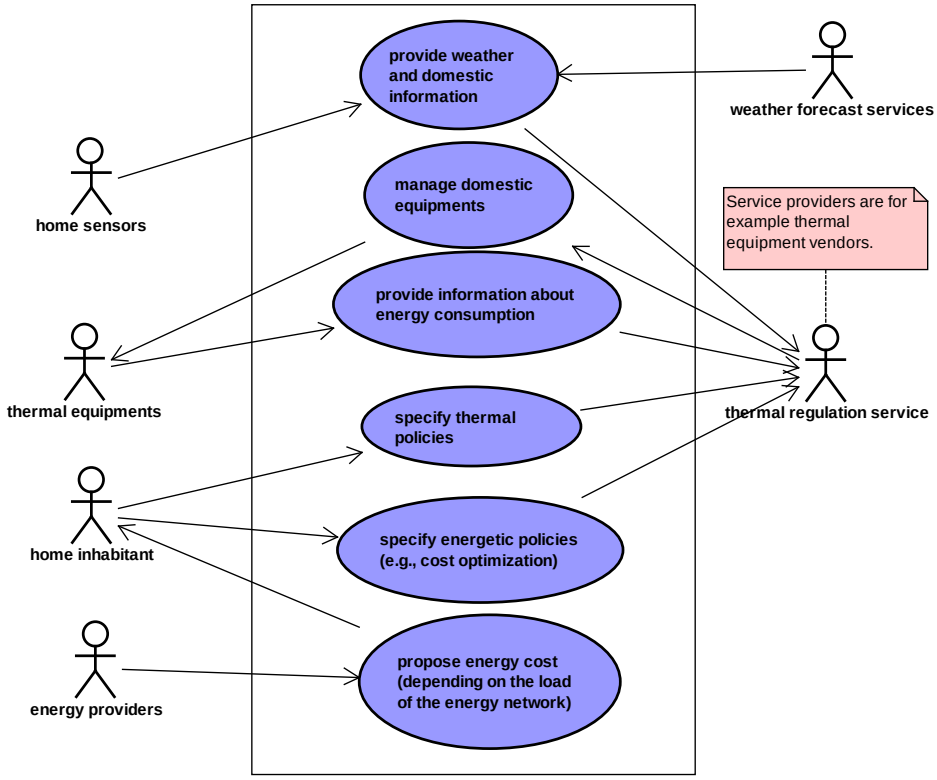


Figure 11: Thermal regulation service context diagram.

Thermal regulation Thermal regulation service providers (typically thermal equipment vendors) are able to manage domestic equipments to achieve thermal regulation specified by policies. Policies are based on goals to be reached such as maximum amounts or acceptable ranges of power consumption. To make regulation decisions, data are collected from domestic environments - and also possibly from the weather forecast service. As different levels of weather forecasts exist in the system (regional, national), the thermal regulation service can subscribe to the one(s) that is(are) relevant.

Thermal regulation exhibits an autonomic behaviour since it implies sending commands to thermal equipments as a reaction to changes in observed thermal conditions. Commands are functions of the delta of in-house temperatures between the observed temperatures and the desired ones. Commands are specific to the target equipements. A thermal regulation service provider may or may not be able to build the proper command to send. Both situations are depicted in Figure 14. Orange node 1 represents a node able to receive data from a house and to send commands to it. Orange node 3 is able to receive data and send commands to house E but not to house D. Commands to house D have to be send through orange node 2. In this case, orange node 3 and 2 have to collaborate (i.e. node 3 will build and send comands to house D on behalf of node 2).

Finally, in thermal regulation phases, the way the energy will be consumed has to be managed to avoid to overload the energy distribution network. Thus, thermal regulation services (from different providers) have to collaborate, as much as possi-

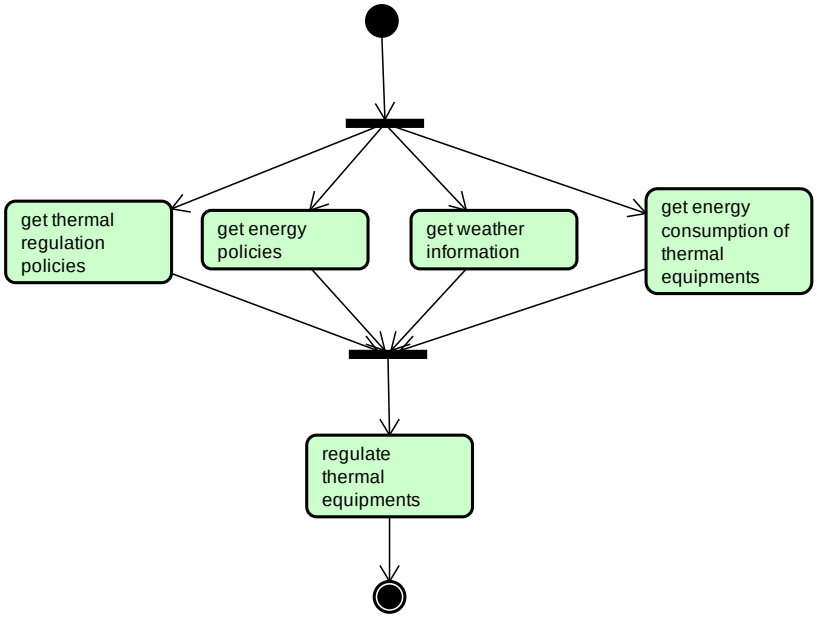


Figure 12: Thermal regulation service activity diagram.

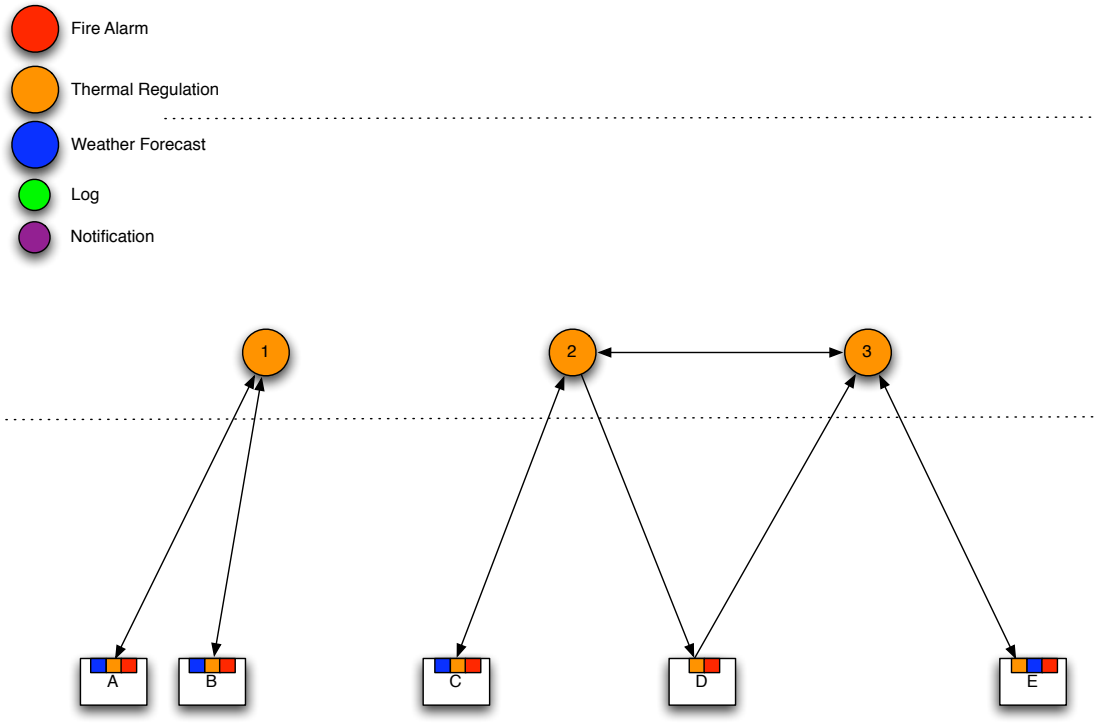


Figure 13: Thermal regulation service architecture.

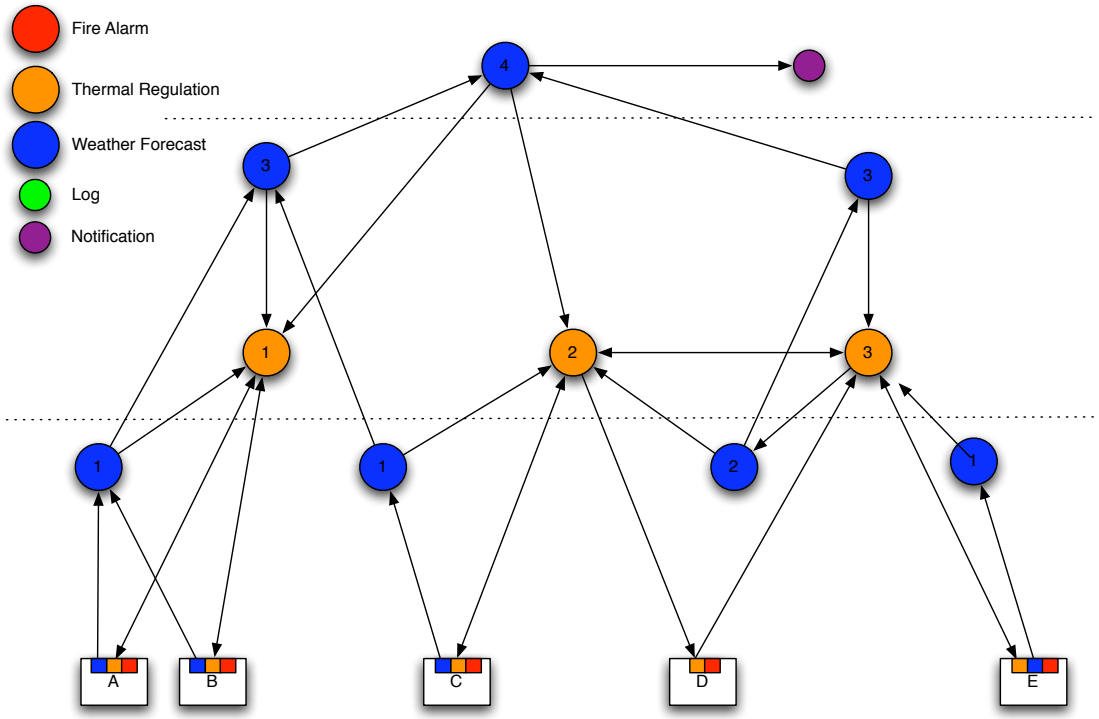


Figure 14: Thermal regulation use case.

ble, to smooth the energetic demands from the houses. This involves collaboration between thermal regulation service providers to plan the commands to send, and with the energy providers to advertise the energetic requirements.

Thermal regulation use data from multiple sources: from sensors (e.g. temperature), from the weather forecast service, from the end-users (e.g. desired thermal ambiance) and from thermal equipments (e.g. power level). Data produced by thermal equipments are pushed at agiven frequency. When a thermal regulation action is engaged, a timeout is set to prevent to send new commands before the effects of the first have an effect (hysteresis). Data from houses, as well as commands, have a time-to-live to avoid unrelevant data or commands.

Logging service The logging service is a classical, generic and common facility share by all M2M services that provide for interfaces used to manage (creation, destruction) and actually use (read and write queries) logs (i.e. traces of services behaviour).

In our M2M use case, the logging service could be implemented as an (P2P) overlay network[3]. When a M2M service node (or client node) needs to access to the logging service, it just has to find a reference to one logging node and then to bind to it. Since all nodes of the logging service are functionally equivalent, a client node can link to any logging node. Log queries are handled by the overlay, i.e. a

[3]This possibility will be further investigated according interest from other Selfman Working Packages.

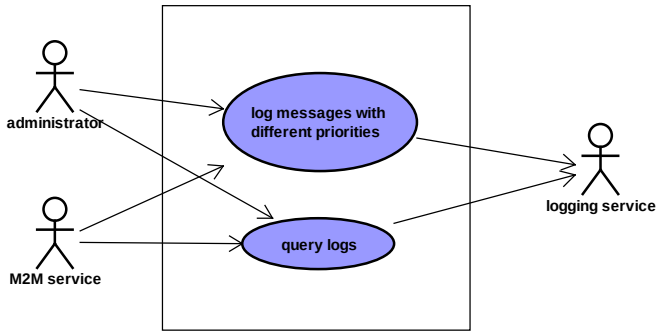


Figure 15: Logging service context diagram.

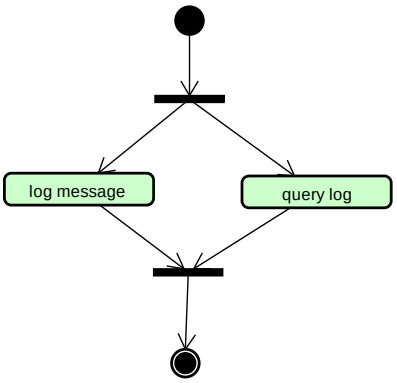


Figure 16: Logging service activity diagram.

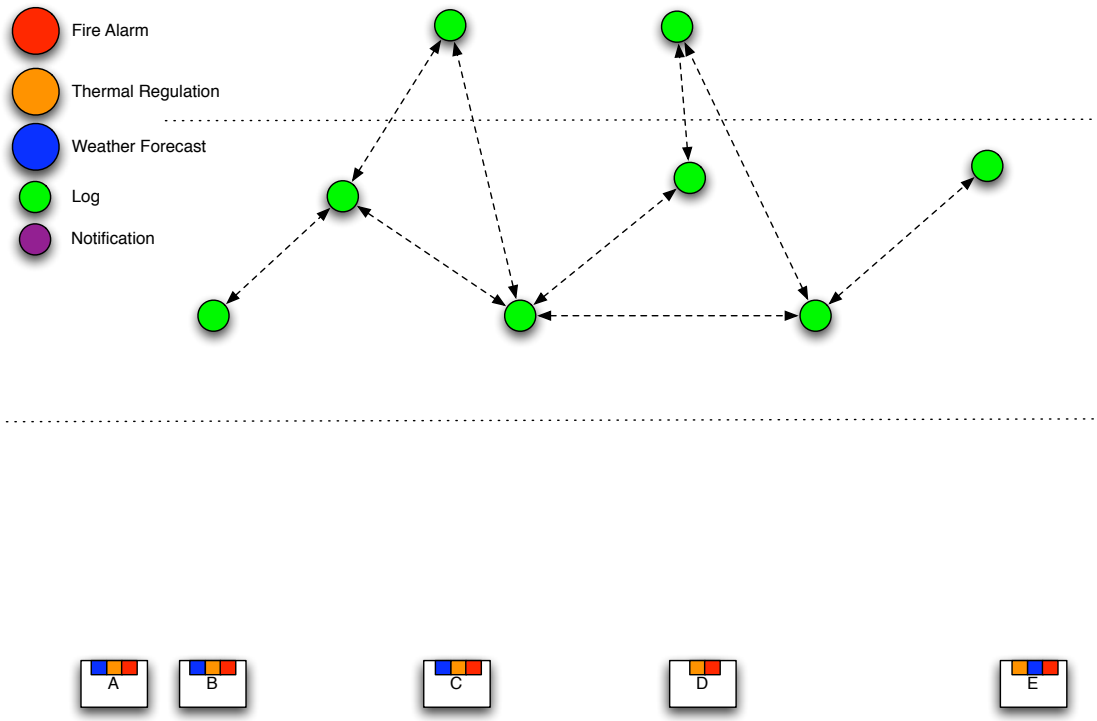


Figure 17: Logging service architecture.

query can be adressed to any logging node, queries and query results are routed by the overlay.

Since logging data can have different priority levels, a QoS level associated to each link between an M2M service node and a logging node. A specific link has then to be established for each level of log needed by the node.

A transactional logging service could also be envisionned with concurrent queries and log persistence[4].

Notification service The notification service is generic, common technical service (enabler) that can notify messages to subscribers on different media: SMS, MMS, e-mail, fax, text-to-speak etc.

As the logging service, the notification service could be architectured in as an overlay[4].

[4]This possibility will be further investigated according interest from other Selfman Working Packages.

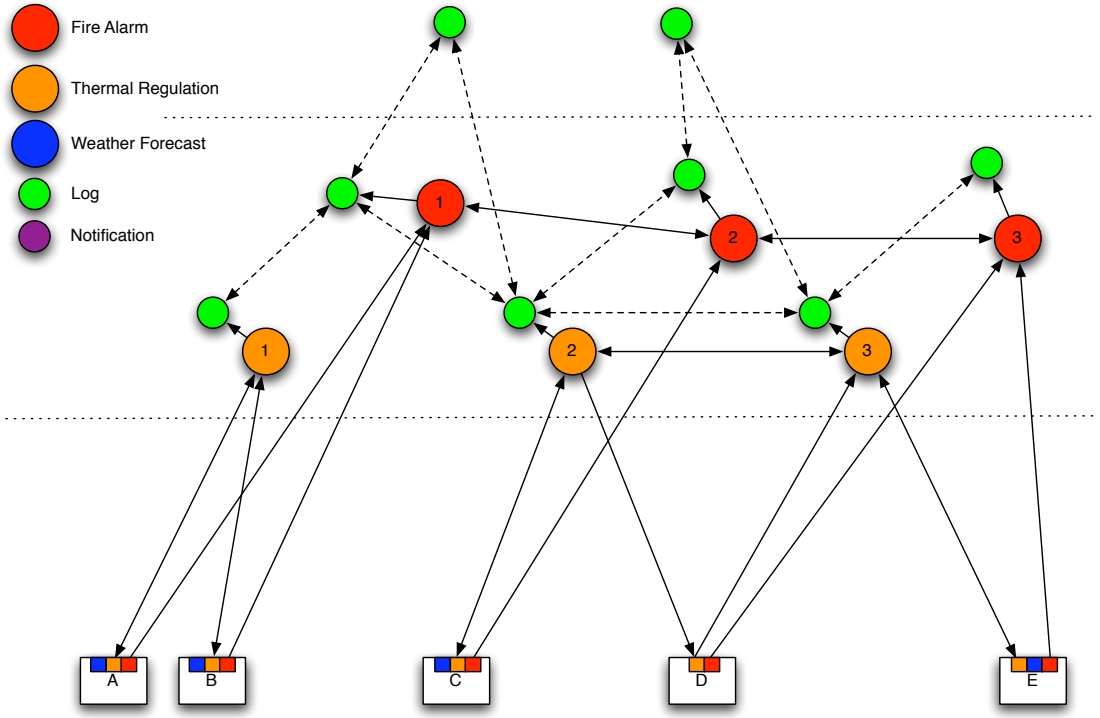


Figure 18: Logging service architecture.

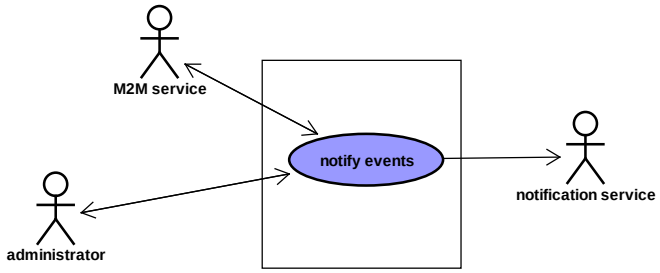


Figure 19: Notification service context diagram.

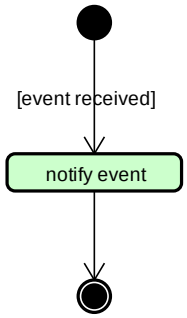


Figure 20: Notification service activity diagram.

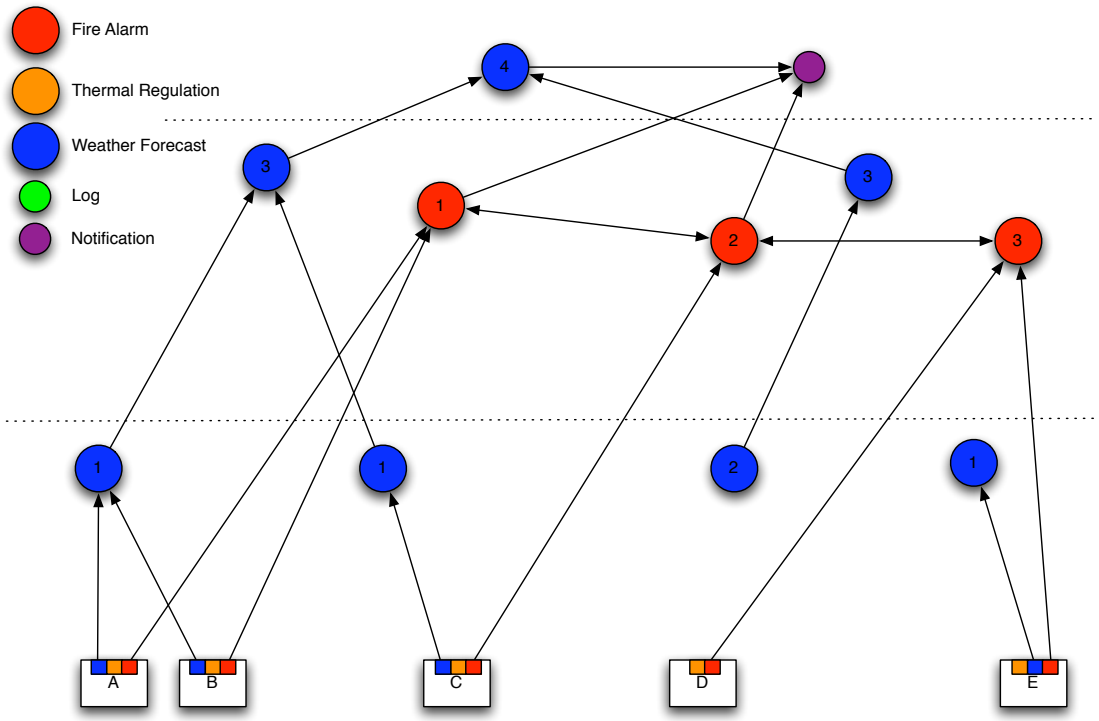


Figure 21: Notification service architecture.

3.2.2 Use Cases

An M2M application, such as presented in the previous section, typically consists of a distributed set of data processing nodes linked together in an arbitrary graph/network. Each node consumes messages that contain data and that come from an arbitrary number of incoming links, then process data in a so-called *proceed* operation implementing some business process, and then produce an arbitrary number of messages containing data on a number of outgoing links.

Such a distributed infrastructure is a typical target for autonomic computing, especially in our M2M use case where the nodes are really numerous (possibly millions of homes), heterogeneous in terms of functionality and criticality, and widely distributed on an arbitrarily complex network topology. An autonomic M2M infrastructure may dynamically and autonomously evolve by means of reconfigurations (adding, removing, updating of nodes and links) in order to adapt to changing execution conditions.

This section specifies the 9 use cases specified in the sequel and listed in the table 1 which indicates the autonomic property each use case concerns among self-repair, self-optimizayion, self-configuration and self-protection as well as the page where the use case is to be found in the document.

Table 1: M2M Uses Cases and typology.

Use Case	Type	Page
UC.MSM2M.GracefullServiceDegradation	Self-Repair	24
UC.MSM2M.NodeFailureRecovery	Self-Repair	27
UC.MSM2M.NodeFaultPrevention	Self-Optimization	30
UC.MSM2M.QoSManagement	Self-Optimization	33
UC.MSM2M.ResourceConsumptionOptimization	Self-Optimization	36
UC.MSM2M.AvailabilityFromRoutClustLoadBal	Self-Optimization	39
UC.MSM2M.LogOverlaySelfConfiguration	Self-Optimization	42
UC.MSM2M.DeploymentOverlaySelfConfiguration	Self-Configuration	45
UC.MSM2M.QualityOfContextManagement	Self-Protection	**??**

Use Case: UC.MSM2M.GracefullServiceDegradation

Scope: M2M Middleware, M2M Multi Service Application (Weather Forecast, Thermal Regulation, Fire Alarming), Self-optimization, Selfman middleware: components

Description: An M2M infrastructure typically hosts several services that are likely to share a number of computing and networking resources. When a serious overload occurs in the M2M infrastructure, a possible solution to prevent a global crash of the infrastructure and hosted services is to apply priority policies between services. Low priority services may be stopped or disconnected in order to reaffect computing and networking resources to the most critical services.

A example of a progressive degradation could be made, starting by disconnecting the weather forecast service, then the thermal regulation service, to keep all resources for the fire alarm service working. Not only some communication links can be disabled in the M2M infrastructure to save network resources, but nodes' hosts may be affected to the fire alarm service.

Here, the autonomic control ensures most critical services still work. This strong enforcement may be applied for severe overload only. This use case relies on detecting special conditions that could make a failure of a critical service provided by M2M like fire detection. One of overload indicators (bandwidth usage on network equipments or M2M nodes, hosts, available memory in JVMs or hosts,CPU usage on hosts, etc) still growing up. A risk of failure on some critical service like is present.

Primary actor: M2MMiddlewareManager (responsible for the global multi service M2M application)

Stakeholders: M2MServiceManager

Preconditions: All services supported by M2M are running. An overload has been detected while self optimisation is already in action.

Trigger: A serious overload occurs in the M2M infrastructure and some overload indicator has reach new high critical level.

Basic course of event (cf. Figures 22):

1. New high critical level of overload is detected.
2. Detect that self optimisation had been already used and no benefic effect on overload has happend.
3. Build an action plan

3.1 determine the nodes wich contribute into the overload (context = which nodes of the notification service are concerned).

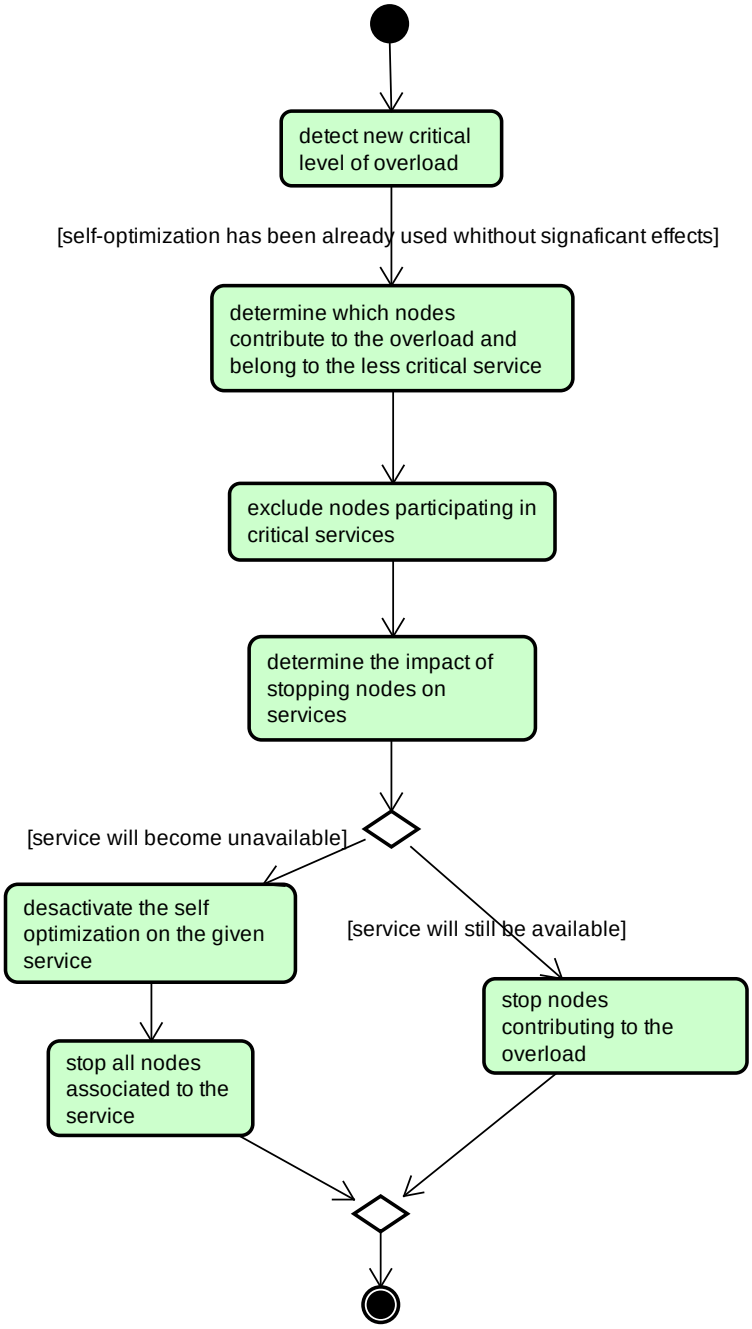


Figure 22: Activity diagram.

3.2 compute nodes sorted by influence on detected overload and belonging to less critical service.

3.3 exclude node which participate on critical service.

3.4 determine impact of stopping node on services.

3.4.1 service will be unavaliable if node is stopped.

3.4.2 desactivate self optimisation on service

3.4.3 stop node and others nodes associated to this service.

4. Notify Service manager

5. Check indicators and go on step 3 or stop use case depending indicators result.

Alternative path: On step 3.4.1 nodes can be stop without impacting whole service. 3.4.2 stop node.

Postconditions: Critical service Fire Alarm is available and fully functional without any failure conditions. Overload indicator decrease significantly.

Miscellaneous:

Use Case: UC.MSM2M.NodeFailureRecovery

Scope: M2M middleware, Selfman middleware: self-repair, components

Description: This scenario consists in reacting when a node has unexpectedly failed. It simply consists ininstantiating a replacement node. The consequence may be different depending onwhether the service is supported by a single node or by a set of nodes. In the formercase, there will be a service disruption, while in the latter case, the service may be maintained, possibly but not necessarily disturbed by a temporary load increase. Common failure detectors are the hello protocol (are you alive?) and the heart-beat protocol (Im alive). Such protocols may be implemented are various levels(e.g. through ICMP or IP network datagrams, JVM monitoring, or via dedicatedmessages between M2M nodes). The higher level, the more reliable the protocol is,because it means all lower levels are OK while you dont know about upper levels.For instance, the network connectivity may be correct while the JVM running theM2M node has crashed. In our use case, this scenario may be applied to the fire alarm service. The service must always be on and quickly responding. So, the autonomic control must ensure that the set of nodes supporting the service is always sufficient. The fire alarm service may temporarily afford a node failure among the set of nodes, whilea new node is being reinstantiated.

Primary actor: M2MMiddlewareManager (responsible for the global multi service M2M application)

Stakeholders:

Preconditions: The nodes are running

Trigger: Failure of a complete M2M node

Basic course of event (cf. Figures 23):

1. detect node failure by failure detection monitoring probes
2. determine failure level
3. recover state of the broken node
4. check if machine that hosted the failed node is available
5. get policy for this node failure recovery
6. build action plan
7. apply action plan including re-instantiate M2M node on the same machine and update incoming/outgoing links
8. get the result status of action plan execution

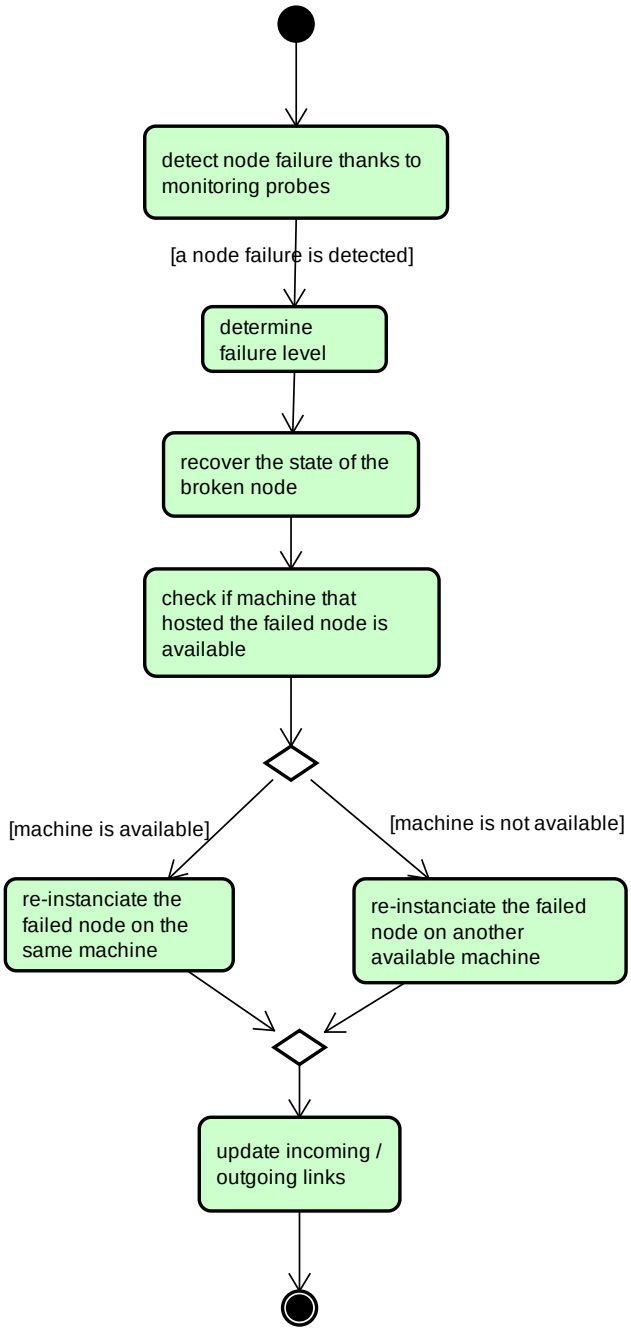


Figure 23: Activity diagram.

8.1 exit if success

8.2 if failure, loop on 6

Alternative Paths: On step 4, if machine hosting failed node is not longer available, re-instantiate M2M node on another available physical node

Postcondition: Failed node has been restarted on same machine or different machine.

Miscellaneous: This use case complements the use case UC.MSM2M.NodeFault-Prevention

Use Case: UC.MSM2M.NodeFaultPrevention

Scope: M2M Middleware (responsible for the notification service), Selfman middleware: self-repair, components

Description: The general use case is about detecting special conditions that are likely to evolve towards a node failure. The specific use case described here concerns node fault prevention in the notification service due to memory leaks. If the available memory on the computer hosting an M2M node is going lower and lower, possibly because of a memory leak in the node implementation, a new node may be instantiated on another host computer or "rebooted" in order to replace the node that is expected to fail.

Primary actor: M2MMiddlewareManager

Stakeholders:

Preconditions: The notification service is running.

Trigger: M2M system has a bad behaviour (memory leak) which can potentially lead to a failure

Basic course of event (cf. Figures 24):

1. Detect failure conditions (here an unsual memory consumption on a given machine)
2. Decide if it corresponds to a memory leak
3. Determine the perimeter of the potential failure (context = which nodes of the notification service are concerned)
4. Build an action plan
5. Test if other machines are available

5.1 instanciate nodes (possibly all) on other available machines

6. transfert state in case of statefull proceed functions
7. stop and discard local nodes
8. start the nodes
9. if failure recover and loop to 4

Alternative path: On step 5.2. reboot nodes locally (node reinstantiation and bindings, state transfer, start the new node)

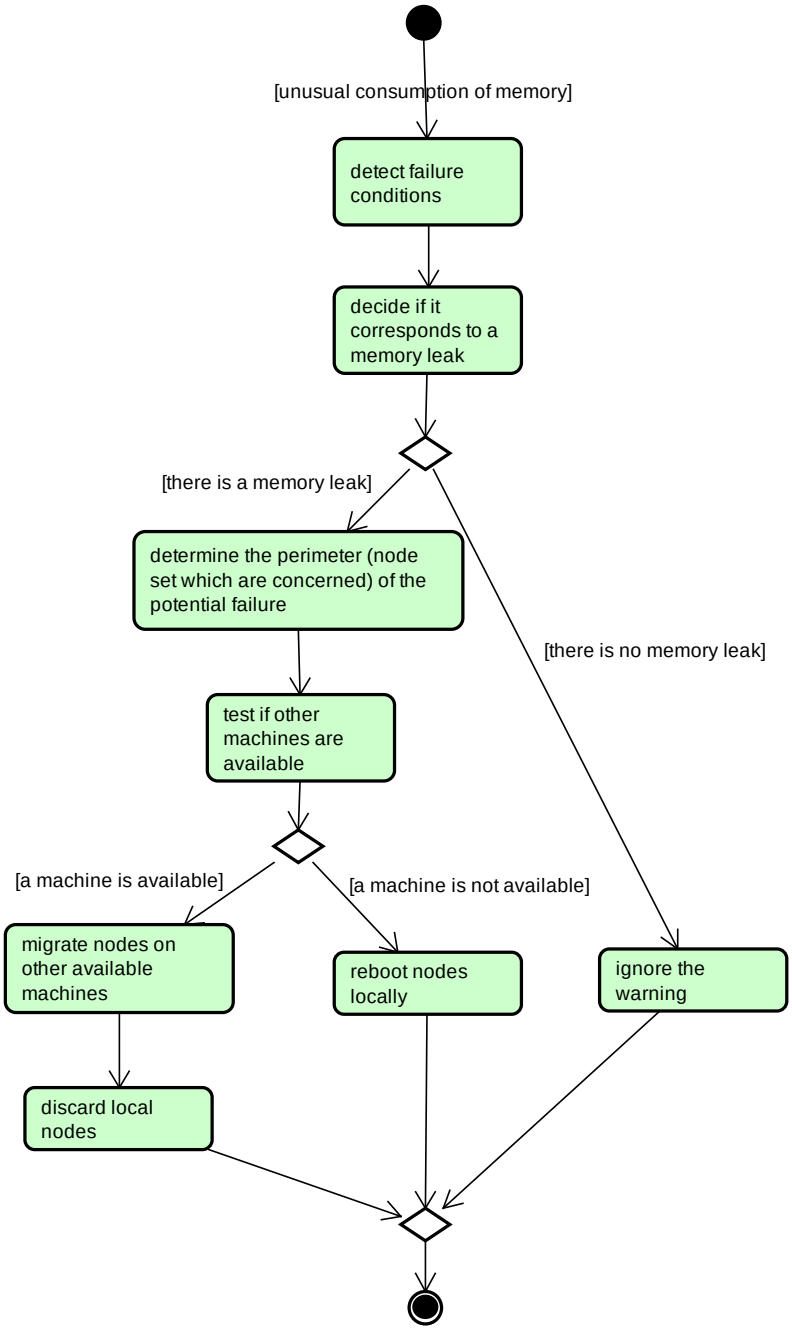


Figure 24: Activity diagram.

Postconditions: The notification service is available and fully functional without any failure conditions.

Miscellaneous:

Use Case: UC.MSM2M.QoSManagement

Scope: M2MMiddleware, Selfman middleware: self-optimisation

Description: The use case consists in managing three kinds of message delivery constraints (aka QoS), so that some messages can be dropped or delayed when a congestion situation occurs in the nodes infrastructure. These QoS are (a) now, (b) now or never, (c) whenever possible. QoS-a typically applies to fire alarms which shall not be dropped nor delayed. QoS-b applies to instant temperature measures for the weather forecast service that may be dropped from time to time but not delayed. QoS-c applies to gas or electricity consumption metering for billing purpose, that may be delayed for a couple of days but not dropped. A dedicated traffic regulator node is introduced in the system to enforce QoS constraints.

Primary actor: M2MMiddlewareManager (responsible for QoS management activation in the M2M multi service application)

Stakeholders:

Preconditions: QoS management is activated and messages with different QoS are being exchanged between nodes.

Trigger: Congestion situation: an input message buffer becomes greater than a given threshold at some node and blocks new incoming messages.

Basic course of event (cf. Figures 25):

1. Output buffers of nodes linked to the congested node also become blocked, and so on in the network of nodes.
2. A traffic regulator service detects that its output message link is blocked due to congestion.
3. The traffic regulator keeps QoS-c messages and don't deliver them anymore
4. The traffic regulator discards QoS-b messages (just some of them, or all of them in critical situations)
5. The traffic regulator sends QoS-a and possibly retained QoS-b messages once its output message link is unblocked.
6. The flow of messages naturally decreases
7. The traffic regulator no longer filters messages and directly pass them.

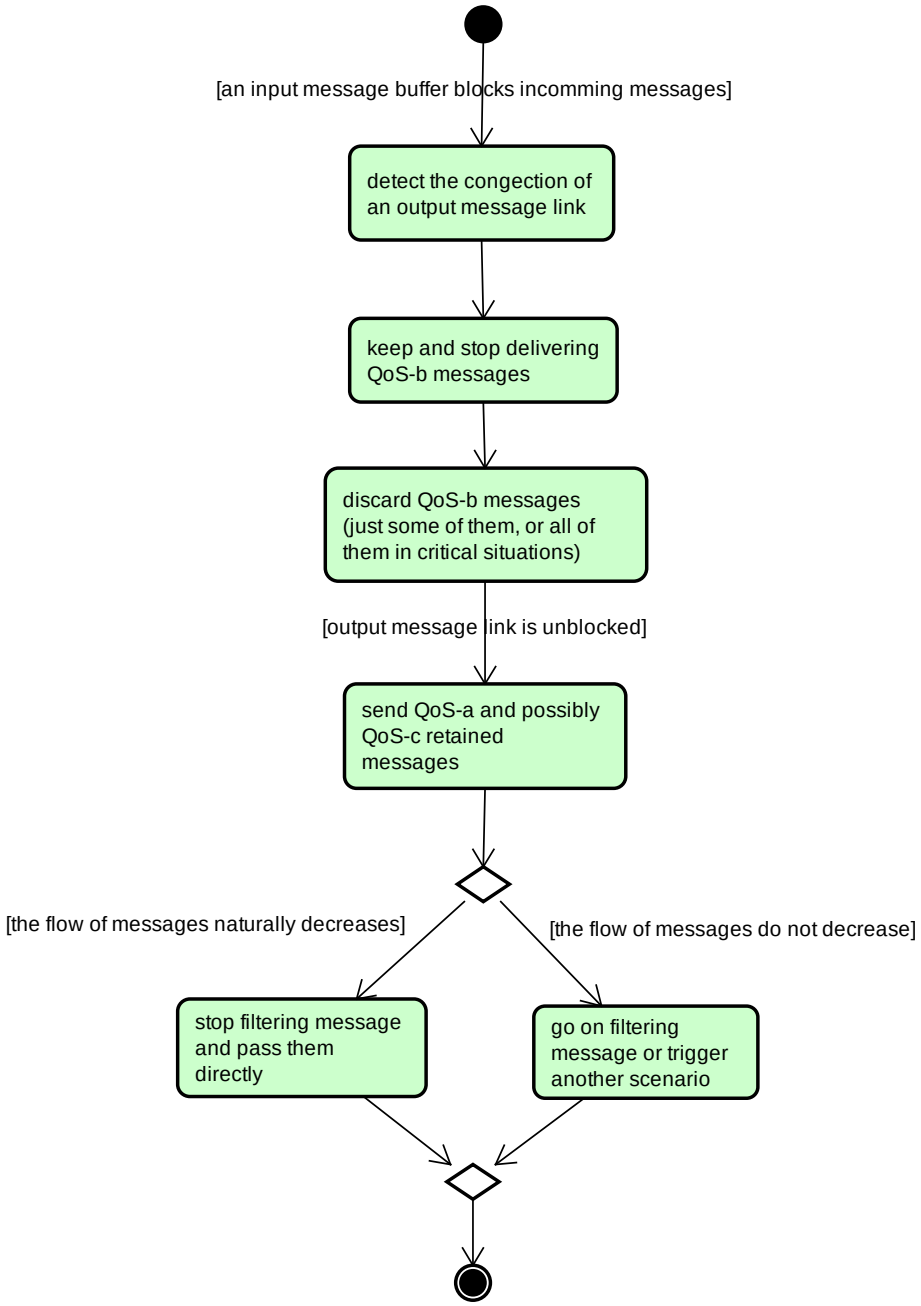


Figure 25: Activity diagram.

Alternative path: At step 6, the flow of messages may not decrease. Then, either the traffic regulator shall go on filtering messages if it is sufficient to solve the congestion problem, or another scenario (cf. UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing) must be triggered either to decrease the flow of important messages (QoS-a and QoS-c) or to increase the processing/networking throughput. This situation can be detected when the number of buffered messages continuously increases and reaches a given threshold.

Postconditions: QoS-a (and possibly some QoS-b) messages have been delivered first, QoS-c messages have been delivered later on, and some QoS-b messages have been discarded.

Miscellaneous: Discarding QoS-b messages shall be achieved in a partial and fair manner: (1) not all QoS-b messages shall be discarded and (2) not all QoS-b messages of a given type and from a given source must be discarded.

Use Case: UC.MSM2M.ResourceConsumptionOptimization

Scope: M2MMiddleware, Selfman middleware: self-optimization, components

Description: This scenario concerns physical resources (network bandwidth, CPU, memory, etc.) usage optimization. When several nodes are executed on physically distributed hosts and are linked to one another to exchange messages, it may be smart to change the configuration, when possible, in order to co-locate them and optimize communication. This may also enable saving power by switching unused hosts off. The links topology will be updated and their nature possibly changed for the sake of efficiency. Some network-supported links would typically be changed into shared memory-based links. The autonomic decision process will have to check that all necessary resources are available at the target host, and that no compatibility or physical attachment constraint forbids the reconfiguration plan.

Primary actor: M2MMiddlewareManager

Stakeholders:

Preconditions: The system is running nominally (the use case is not triggered if system is already in a self-repair use case)

Trigger: The M2MMiddlewareManager decides to optimize resources consumption.

Basic course of event (cf. Figures 26):

1. determine available resources cartography (physical location and links, etc.)
2. determine available nodes cartography (geographical location, logical links, etc.)
3. determine individual physical resources (CPU, memory, etc.) usage
4. determine available physical resources (CPU frequency, memory size, free disk space, etc.).
5. determine intensive CPU and bandwidth usage nodes (physical links)
6. select M2M nodes to migrate
7. check possible physical attachment to devices and compatibility constraints in the reconfiguration plan
8. deactivate intensive resources usage nodes (save their states)
9. choose according to step 7 constraints the physical node(s) where to migrate node(s)

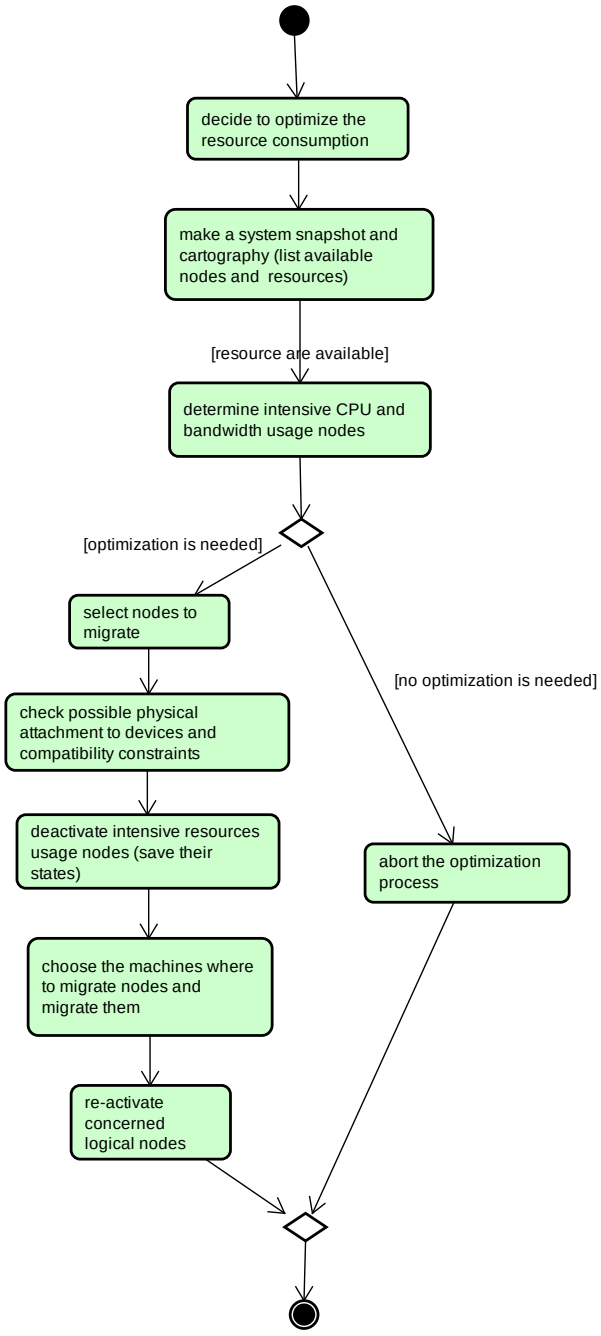


Figure 26: Activity diagram.

10. migrate node(s) to chosen machines.
11. re-activate concerned logical node(s).

Alternative path:

- On step 4, if there is no free computing resources, wait for resources availability
- On step5, if there is no intensive bandwith usage nodes, abort the optimization process
- On step6/7, if selected nodes are no longer available, abort the optimization process

Postconditions: Resources usage is optimized

Miscellaneous:

Use Case: UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing

Scope: M2MMiddleware, Selfman middleware: self-optimisation

Description: The use case consists in managing the availability of a communication path (i.e. a list of nodes and links engaged in) between one or many source nodes and one or many destination nodes. To reach that goal, three mechanisms will be used: routing from node to node; using node clustering to cope with node failures; load balancing to deal with over-loading on dedicated path. The M2M-middleware will have to combine all of them to ensure communications regarding the overall system state.

Primary actor: M2MMiddlewareManager conceives the need of infrastructure for load balancing, routing and clustering

Stakeholders: none

Trigger: At least one hope in the communication path is impossible or do not feet with the non-functional requirements (cf. UC.QosManagement).

Basic course of event (cf. Figures 27):

1. a message reaches a step from which the communication is impossible forward.

2. the context of the impossibility has to be defined

3.1.1. an alternative path is available

3.1.2. the message is routed thanks to the alternative path

3.2.1. it's possible to create equivalente nodes

3.2.2 a cluster of at least two nodes is created

3.2.3. the message is treated by a node of the cluster

3.3.1. it's possible to create equivalente nodes and a policy is set to choose one

3.3.2. a set of at least two nodes is created

3.3.3. the message is treated by the node designated thanks to a give, policy

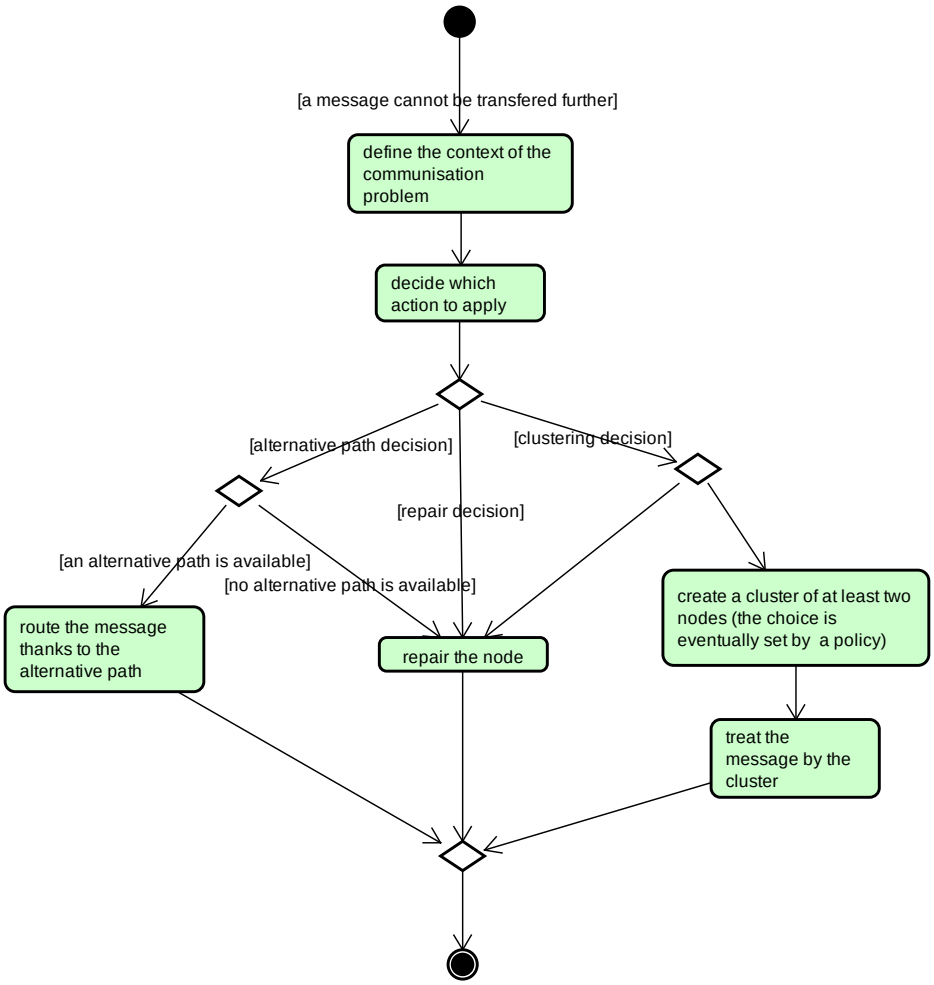


Figure 27: Activity diagram.

Alternative path:

3.1.1 None alternative path is found. The failed node has to be repaired. (cf. UC.MultiServicesM2M.NodeFailureRecovery)

3.2.1 It is not possible to create equivalente nodes. The failed node has to be repaired. (cf. UC.MSM2M.NodeFailureRecovery)

3.3.1 It is not possible to create equivalente nodes. The failed node has to be repaired. (cf. UC.MSM2M.NodeFailureRecovery)

3.3.3 The policy is unable to designate a node from the set to treat the message. The message has to wait for a node to be available.

Postconditions: The message reaches the destination(s).

Miscellaneous:

Use Case: UC.MSM2M.LogOverlaySelfConfiguration

Scope: M2MMiddleware, Selfman middleware: overlays, self-configuration

Description: The use case consists in deploying an M2M technical service, and for specifically the logging functionnality of the M2M application as an overlay network (P2P system). M2M application itself does not seem particularly suited to a implementation as overlay because all M2M nodes do have a specific behaviour. On the contrary, non functionnal or technical services of the M2M application seems more adapted since all nodes in an overlay would provide the same functionnality (e.g. deployment, persistance, etc.).

The approach would be applied here to the log service. This service provides a number of other services (weather forecast, thermal regulation, fire alarm) with a persistence facility for events that may be useful to consult some time later (e.g. for troubleshooting purpose). Peer log nodes would be linked with each other automatically in a self-configured topology.

Primary actor: M2MMiddlewareManager (responsible for non functional properties of the M2M multi service application)

Stakeholders: M2MServiceManager

Preconditions: No logging service is running in the system

Trigger: A M2M service needs a logging service and asks the M2MMiddlewareManager to deploy it for M2M nodes

Basic course of event (cf. Figures 28):

1. determine available machines on which the logging overlay can be deployed (maximum threshold)
2. determine number of nodes needed for the log overlay based on the number of nodes that will use the log service (minimum threshold)
3. deploy (install) the log overlay peers on determined nodes
4. self-configure the log overlay based on storage capability of each log peer
5. connect nodes with the log overlay (each node is connected to its closest node)
6. self-configure the overlay dynamically to adapt to logged nodes appearance/disappearance

Alternative path: On step 2, if minimum threshold, maximum threshold then deployment of log overlay is not possible (the use case stops)

Postconditions: An logging overlay is available

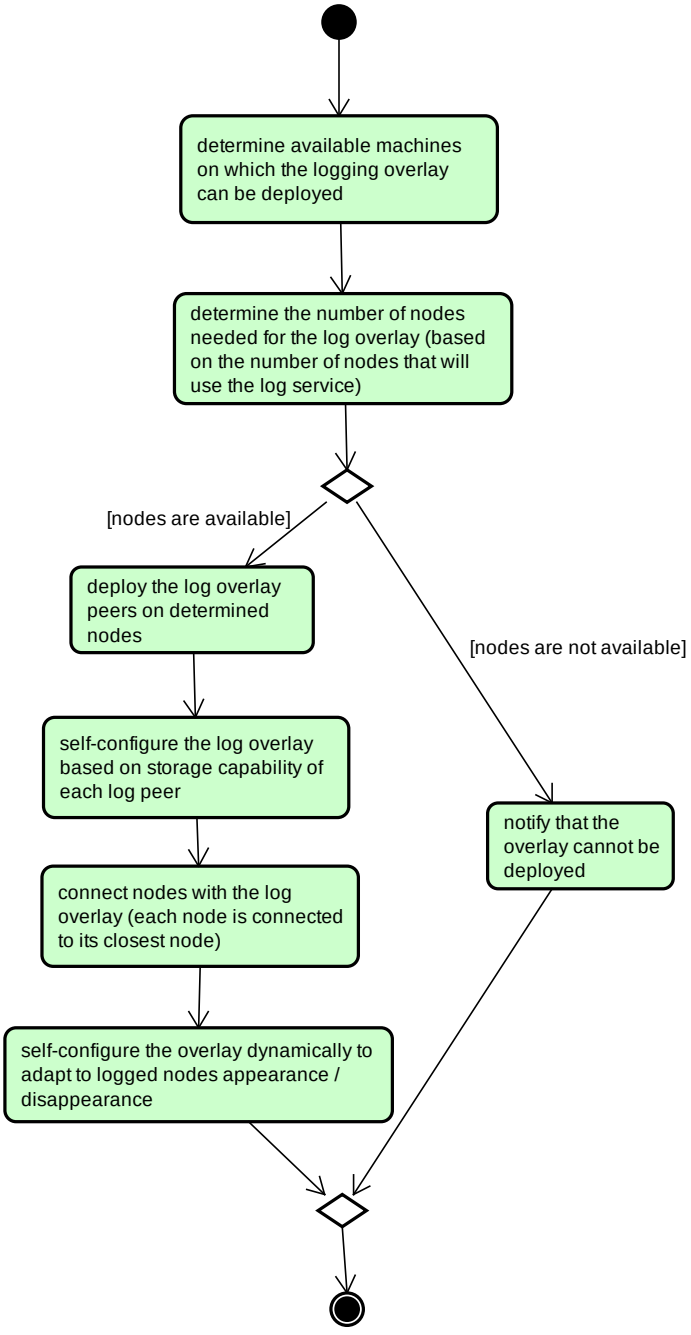


Figure 28: Activity diagram.

Miscellaneous: The use case can be declined for other non functional (infrastructures services, technical services) such as persistance, notification or deployment. By elarging the vision, this use case might prefigure distributed containers on overlays: one can imagine for instance, a J2EE server implemented as a set of interacting overlays.

Use Case: UC.MSM2M.DeploymentOverlaySelfConfiguration

Scope: M2MMiddleware, Selfman middleware: overlays, self-configuration

Description: The use case consists in deploying an M2M technical service, and for specifically the deployment functionnality of the M2M application as an overlay network (P2P system). M2M application itself does not seem particularly suited to a implementation as overlay because all M2M nodes do have a specific behaviour. On the contrary, non functionnal or technical services of the M2M application seems more adapted since all nodes in an overlay would provide the same functionnality (e.g. deployment, persistance, etc.).

The approach would be applied here to a deployment service that would be used to deploy an M2M multi service application. Peer log nodes would be linked with each other automatically in a self-configured topology.

Primary actor: M2MMiddlewareManager (responsible for non functional properties of the M2M multi service application)

Stakeholders: M2MServiceManager

Preconditions: No logging service is running in the system

Trigger: A M2M service needs a logging service and asks the M2MMiddlewareManager to deploy it for M2M nodes

Basic course of event:

1. determine available machines on which the logging overlay can be deployed (maximum threshold)
2. determine number of nodes needed for the log overlay based on the number of nodes that will use the log service (minimum threshold)
3. deploy (install) the log overlay peers on determined nodes
4. self-configure the log overlay based on storage capability of each log peer
5. connect nodes with the log overlay (each node is connected to its closest node)
6. self-configure the overlay dynamically to adapt to logged nodes appearance/disappearance

Alternative path: On step 2, if minimum threshold, maximum threshold then deployment of log overlay is not possible (the use case stops)

Postconditions: An logging overlay is available

Miscellaneous: The use case can be declined for other non functional (infrastructures services, technical services) such as persistance, notification or deployment. By elarging the vision, this use case might prefigure distributed containers on overlays: one can imagine for instance, a J2EE server implemented as a set of interacting overlays.

Use Case: UC.MSM2M.QualityOfContextManagement

Scope: M2MMiddleware.

Description: The use case consists in reacting to a change in the quality of the context (Qoc) information transmitted in the M2M infrastructure. By QoC, we refer to the different dimensions related to the confidence one can have in information describing the current applicative context, such as security, privacy, safety, reliability, or precision. These dimensions can be described by augmenting context information with meta-data processed throughout the steps of acquisition, aggregation/fusion, and exploitation of context information by applications. Monitoring the QoC level by performing a thorough confidence assessment, and providing QoC guarantees become major requirements for contextual adaptations of safety-critical applications such as fire alarm detection, or temperature regulation, in which error-prone/malicious interpretation (e.g., false positive/false negative) of unreliable context data can result in disaster situations.

In this use-case, we consider a single dimension of QoC, the reliability of alerts produced by a smoke detector, measured by a scalar confidence parameter, e.g., the probability of false positive detections. In the current situation, a smoke detector was identified as having a false positive rate greater than a specified threshold. This rate can typically be calculated by correlating the number of fire alerts raised by this detector to the number of real fire situations in the concerned house (for instance, as reported by fire brigades). All smoke detectors and other nodes of the M2M infrastructure exhibit their QoC level (provided and required) through contracts, the satisfaction/violation of which is monitored by the M2M midleware.

Primary actor: SmokeDectector.

Stakeholders: FireAlarm Service, M2MMiddlewareManager (responsible for QoC management).

Preconditions: The QoC management service is activated. Messages with different QoC are being exchanged between the smoke detectors and the fire alarm service through intermediary M2M nodes. QoC contracts have been defined for each smoke detector, middle nodes, and for the fire alarm service.

Trigger: The QoC management service detects a violation of a QoC contract.

Basic course of event

1. Assess if the violated QoC contract is linked to smoke detection and fire alarm. If not, exit the mis-use case.

2. Identify the cause of the QoC contract violation. If the cause if not related to reliabity (as measured by the probability of false fire alerts), exit the mis-use case.

3. Downgrade the reputation of the smoke detector according to reputation model chosen beforehand, for instance by black-listing the smoke detector behaving suspiciously.

4. Cut the connection from the smoke detector to the first node in the M2M infrastructure it is connected to.

5. Raise an alert to signal that the concerned house has no longer any fire alarm detection.

Alternative path: At step 3, if other smoke detectors are available in the concerned house that fulfill the QoC contract, choose the smoke detector with the best reputation. Subsequently, activate the connection from the choosen smoke detector to the M2M infrastructure, and cut the connection from the faulty smoke detector to the M2M infrastructure.

Postconditions: The faulty smoke detector has been disconnected from the M2M system or replaced by another one with a greater QoC.

Miscellaneous: This mis-use case concerns a reliability-oriented QoC (as measured by a false positive fire alarm rate). Similar mis-use cases can also be described for other QoC dimensions (possibly used by other M2M services) such as authenticity, confidentiality, integrity, precision, or privacy of context information. Such degradation of QoC may reflect a variety of attacks such as sensor (resp. M2M node) spoofing in the M2M service layer, in which a malicious (e.g., with fake identity) sensor (resp. M2M node) sends/floods the rest of the M2M system with corrupt/garbage data, or hampers routing (e.g., critical packets are dropped).

3.2.3 Requirements

This section identifies the requirements associated the autonomic scenarios in the considered M2M applicative context presented in the previous sections. The section is organized in two parts. This first ones identifies requirements in terms of functional and non functional desired properties. The second ones indentifies operational requirements which are to be understood as requirements on the Selfman middleware (components, overlays, self-* mechanisms, transactions). They are to be taken as inputs fot Selfman WPs 1-4.

The following table lists the requirements, their type (properties or mechanisms) and pages where they can be found in the sequel.

Requirement ID	Type	Page
RQ.MSM2M.Manageability	Property	50
RQ.MSM2M.Reliability	Property	51
RQ.MSM2M.Availability	Property	52
RQ.MSM2M.Scalability	Property	53
RQ.MSM2M.DeploymentOverlayScalability	Property	54
RQ.MSM2M.RQ.MSM2M.LogOverlaySelfConfigurability	Property	55
RQ.MSM2M.RQ.MSM2M.DeploymentOverlaySelfConfigurability	Property	56
RQ.MSM2M.Observability	Property	57
RQ.MSM2M.Accountability	Property	58
RQ.MSM2M.Autonomicity	Property	59
RQ.MSM2M.Components	Mechanism	60
RQ.MSM2M.AutonomicDecision	Mechanism	61
RQ.MSM2M.AutonomicSimulation	Mechanism	63
RQ.MSM2M.TransactionalReconfiguration	Mechanism	64
RQ.MSM2M.TransactionalIntraNodeProcessing	Mechanism	66
RQ.MSM2M.TransactionalLogging	Mechanism	67
RQ.MSM2M.TransactionalDeploymentOverlay	Mechanism	68
RQ.MSM2M.ReputationManagement	Mechanism	69

Table 2: Synthesis of operational requirements of the 4 Selfman applications on WP1-Overlays, WP2-Components, WP3-Transactions and WP4-Self-* Services.

RequirementID: RQ.MSM2M.Manageability

Use case(s): UC.MSM2M.NodeFaultPrevention, UC.MSM2M.NodeFailureRecovery, UC.MSM2M.QoSManagement, UC.MSM2M.ResourceConsumptionOptimization, UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing UC.MSM2M.QualityOfContextManagement

Priority: Must

Description: Manageability is to be taken here in a broad meaning, especially encompassing flexibility, modularity (composability) and maintainability. Manageability refers here to the ability to configure and dynamically reconfigure homogeneously both individual components and component assemblies forming an M2M system.

Rationale: In order to enhance an M2M infrastructure with autonomic features, it must be possible to easily manipulate complex software configurations in terms of components, connections between components (i.e. M2M nodes and links) and M2M nodes internals (sub-components). Since autonomic features require runtime adaptations, configuration and dynamic reconfigurations should both be supported.

Type: Non functional property

Dependency: RQ.MSM2M.Observability

Assessment criteria: The requirement assessment would be qualitative and reported in D5.4:

- check availability of a component programming framework in Selfman middleware that support configuration and dynamic reconfiguration
- check availability of M2M component-based middleware by means of re-engineering of M2M nodes or wrapping of M2M nodes as components

Target: WP2 components

RequirementID: RQ.MSM2M.Reliability

Use case(s): UC.MSM2M.NodeFaultPrevention, UC.MSM2M.NodeFailureRecovery

Priority: Should

Description: This desired property encompasses reliabilty and integrity. It refers here to the ability to guarantee a correct behaviour of a system in a given environment. The reliability property is assessed against its integrity supposedly defined by integrity constraints on the state and behaviour of a system. In our M2M context, some integrity constraints, such as the ones that concern reliable data routing and processing (e.g. no data must be lost, all data must be processed), may be considered as functional. Some others may be considered as non functional: typically the ones that concern the reliable (transactional) reconfigurations of an M2M infrastructure.

Rationale: Reliability is a strong requirement in M2M systems. Autonomic features must of course enforce reliability.

Type: Non functional property

Dependency: RQ.MSM2M.Manageability, RQ.MSM2M.Observability

Assessment: The requirement assessment would be quantitative and reported in D5.5. Scenarios would be defined and executed by a traffic generation (load injection) tool to test functional and non functional reliability. For non functional reliability (integrity) which is the one we are more interested in, scenarios would include the execution of erroneous/invalid reconfigurations. A successful support of the requirement would be that the M2M system detects and prevents these erroneous reconfigurations and remains in consistent state.

Target: WP2 components, WP4 self-repair, WP1 overlays

RequirementID: RQ.MSM2M.Availability

Use case(s): UC.MSM2M.NodeFailureRecovery

Priority: May

Description: This desired property refers to the ability of a system to ensure its function the maximum possible amount of time. Availability is generally seen as a ratio between Mean time between failure (MTBF) and Mean time to repair (MTTR).

Rationale: M2Msystems we consider put serious requirements of the availability of the supporting M2M infrastructure especially in presence of different quality of service (QoS) on data routing and processing, and priorities between M2M services.

Type: Non functional property

Dependency: RQ.MSM2M.Manageability, RQ.MSM2M.Observability

Assessment: The requirement assessment would be quantitative and reported in D5.5. Scenarios would be defined and executed by a load injection tool. The scenarios would be based on fault injectio n: faults provoking node failures (e.g. by simply killing the process executing a JVM hosting a M2M node) would be injecting on various rates to simulate sample MTBF and then corresponfing MTTR will be measured. A successful support of the requirement would be that the M2M system exhibits an average ratio between MTBF and MTTR above a threshold (that remains to be determined in D5.5).

Target: WP2 components, WP4 self-repair

RequirementID: RQ.MSM2M.Scalability

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.GracefulServiceDegradation,
UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing

Priority: May

Description: This desired property refers to the ability to either handle growing amounts of work, in a graceful manner, or to be readily enlarged.

Rationale: In our M2Mcontext, this feature refers i) to the ability to support the equipments of thousands to millions of domestic environments and ii) to support the addition of new M2Mservices (or applications e.g. Thermal regulation, fire alarming, weather forecast, logging, notification in our applicative context). Both generally in a growing number of M2M nodes and links.

Type: Non functional property

Dependency: RQ.MSM2M.Manageability, RQ.MSM2M.Observability, RQ.-MSM2M.Accountability, RQ.MSM2M.Availability, RQ.MultiServicesM2M

Assessment: The requirement assessment would be quantitative and reported in D5.5. Scenarios would be defined and executed by a load injector tool. The scenarios would represent a few configurations of a M2M system representative of an order of magnitude (e.g. 10, 100 and 1000 nodes) by varying the number of node representing houses only so as to keep the rest of the M2M system constant. Average performance would be measured and compared on these representative configurations. A successful support of the requirement would be that the M2M system performance on these various configurations in at least one selected use case remains inside a given interval (that remains to be determined in D5.5).

Target: WP5 M2M middleware, WP4 self-optimization

RequirementID: RQ.MSM2M.DeploymentOverlayScalability

Use case(s): UC.MSM2M.DeploymentOverlaySelfConfiguration

Priority: May (the M2M application may support support this requirement but it is optional)

Description: Scalability if the deployment process on overlay networks refers to the capacity of the M2M system to handle growing numbers of M2M nodes to be deployed without losing significant performance.

Rationale: In the M2M application, this feature refers i) to the ability to support the deployment of huge number of M2M nodes especially M2M node inside thousands to millions homes.

Type: Non functional property

Dependency: RQ.MSM2M.Manageability, RQ.MSM2M.Observability, RQ.-MMSM2M.Accountability, RQ.MSM2M.Availability, RQ.MultiServicesM2M

Assessment: The requirement assessment would be quantitative and reported in D5.5. The test scenario would, by means of a deployment overlay network, deploy a M2M system of different orders of magnitude (eg 10, 100 and 1000 nodes). Average time to perform complete deployment would be measured and compared on these representative configurations. A successful support of the requirement would be that the M2M system performance on these various configurations remains inside a given interval (that remains to be determined in D5.5).

Target: WP5 M2M middleware, WP4 self-optimization

Requirement: RQ.MSM2M.LogOverlaySelfConfigurability

Use case(s): UC.MSM2M.LogOverlaySelfConfiguration

Priority: May

Description: For the use cases UC.MSM2M.LogOverlaySe lfConfiguraton and to be supported, a log service as an overlay must be availab le in Selfman. This can typically be built on the DDB Selfman middleware (logging may be seen here as a basic form of persistence).

Rationale: Overlay networks naturally exhibit good properties essentially w.r.t. scalability, availability and autonomicity (mainly self-configuration). The M2M use case, and more specifically the self-configuration of the deployment overaly scenario exhibits a requirement towards self-configuration (the deployment overlay nodes should discover each others, determine and maintain routes/connections between each others automatically) and possibly self-dimensioning (the deployment overlay could determine itself how many deployment nodes with which storage capability it needs according to the number of deployment actions it is asked to perform).

Type: Operational required mechanism (operational/overlay, operational/self-configure)

Dependency:

Assessment: The requirement assessment would be both qualitative and quantitative an and reported in D5.4 and D5.5:

- check availability of logging service as an overlay in Selfman middleware
- check capability to self-configure the log overlay in various settings of node/storage capability i.e. various conditions of minimum/ maximum threshold (scenarios would be defined and executed by a load injector tool)
- check that all logging data is logged in the overlay (no data loss)
- check capability of the log overlay to self-adapt to M2M nodes appearance/ disappearance (scenarios would be defined and executed by a load injection tool).

A successful support of the quantitative requirements (bullets 2 and 4 abo ve) would be that the M2M system manages to keep in a given interval (to be determined in D5.5) the ratio between the amount of data (and secondly of the traffic of log requests (especially read requets) to be logged (in bytes) and the number of nodes in the log overlay.

Target: WP1 overlays, WP5 DDB middleware

Requirement: RQ.MSM2M.DeploymentOverlaySelfConfigurability

Use case(s): UC.MSM2M.DeploymentOverlaySelfConfiguration

Priority: May

Description: For the use cases UC.MSM2M.DeploymentOverlaySelfConfiguration to be supported, a deployment service as an overlay must be available in Selfman. This can typically be built on the DDB Selfman middleware (logging may be seen here as a basic form of persistence).

Rationale: Overlay networks naturally exhibit good properties essentially w.r.t. scalability, availability and autonomicity (mainly self-configuration). The M2M use case, and more specifically the self-configuration of the deployment overlay scenario exhibits a requirement towards self-configuration (the deployment overlay nodes should discover each others, determine and maintain routes/connections between each others automatically) and possibly self-dimensioning (the deployment overlay could determine itself how many deployment nodes and their associated storage capability it needs according deployment queries it receives).

Type: Operational mechanism

Dependency:

Assessment: The requirement assessment would be both qualitative and quantitative and reported in D5.4 and D5.5:

- check availability of a deployment service as an overlay in Selfman middleware
- check capability of the deploement overalyto self-configure in various settings of nodes/storage capability i.e. various conditions of minimum/ maximum threshold (scenarios would be defined and executed by a load injection tool)
- check capability of the deployment overlay to self-adapt to M2M nodes appearance/disappearance (scenarios would be defined and executed by a load injection tool)

A successful support of the quantitative requirements (bullets 2 and 4 above) would be that the M2M system manage to keep in a given interval (to be determined in D5.5) the ratio between the number of deployement actions and the number of nodes in the deployment overlay.

Target: WP1 overlays, WP5 DDB middleware

RequirementID: RQ.MSM2M.Observability

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.GracefulServiceDegradation,
UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing
UC.MSM2M.QualityOfContextManagement

Priority: Must

Description: Observability refers to the capability to easily add/remove probes in an M2M system and more generally to introspect its architecture and behaviour.

Rationale: As an M2M application possibly involves, as in our sample applicative cont ext, multiple actors (people living in the houses, people from the fire department or weather forecast service, etc.) including actors that support critical busines ses (e.g. fire alarming), it is of paramount importance for an M2M infrastruct ure to provide means for easily add, remove, manipulate sensors.

Type: Non functional property

Dependency: none (this a basic/fundamental requirement)

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check availability of a monitoring framework in Selfman middleware
- check availability of probes needed in use cases
- check capacity of inserting needed probes in M2M application

Dependency:

Target: WP2 components

RequirementID: RQ.MSM2M.Accountability

Use case(s): UC.MSM2M.LogOverlaySelfConfiguration

Priority: Should

Description: Accountability refers to the capability to trace data and resources consumption and to associate it to a specific M2M service for billing and responsability management purposes.

Rationale: As an M2M application possibly involves, as in our sample applicative context, multiple actors (people living in the houses, people from the fire department or weather forecast service, etc.) including actors that support critical businesses (e.g. fire alarming), it is of paramount importance for an M2Minfrastructure to be able i) to define responsibilities in case of malfunctions (e.g. dataloss) and failures (e.g.crash of nodes or links) and ii) to assess resource consumption per services in a billing perspective (pay-per-use).

Type: Non functional property

Dependency: RQ.MSM2M.Observability, RQ.MSM2M.LogOverlaySelfConfiguration

Assessment: The requirement assessment would be qualitative and reported in D5.4. The assesment would be limited to data tracing and supported by the log overlay if present. Hence the assessment of this requirements mainly boils down to the assessment of UC.MSM2M.LogOverlaySelfConfiguration.

Target: WP5 M2MMiddleware, WP3 DDB middleware, WP1 overlay,

RequirementID: RQ.MSM2M.Autonomicity

Use case(s): UC.MSM2M.NodeFaultPrevention, UC.MSM2M.NodeFailureRecovery, UC.MSM2M.QoSManagement, UC.MSM2M.ResourceConsumptionOptimization, UC.MSM2M.OverlaySelfConfiguration, UC.MSM2M.GracefulServiceDegradation, UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing UC.MSM2M.QualityOfContextManagement

Priority: Must

Description: This desired property refers to *self-* properties*, i.e. the ability for a system i) to observe its structure, behaviour and environment, and ii) to take corrective actions if needed.

Rationale: Autonomicity is of primary importance in large scale, dynamic, open M2M systems, such as the ones envisioned here, in which a "manual" (by human operators) configuration and management is almost impossible. Autonomic computing seeks fundamentally to automate as much as possible the deployment and management of software systems so as to lessen human interventions and afferent costs - which is very relevant in M2M settings.

Type: Non functional property

Dependency: RQ.MSM2M.Observability, RQ.MSM2M.Manageability

Assessment: The requirement assessment would be qualitative and reported in D5.4. A successful support of the requirement would be that the M2M system supports requirements RQ.MultiServicesM2M.Observability and RQ.MultiServicesM2M.Manageability and at least one autonomic use case and associated requirement(s).

Target: WP2 components, WP4 self-management services

RequirementID: RQ.MSM2M.Components

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.GracefulServiceDegradation,
AvailabilityFromRoutingClusteringLoadBalancing
UC.MSM2M.QualityOfContextManagement

Priority: Should

Description: The Multi Service M2M application demands a sound architectural framework which allows for M2M system configuration, deployment and management. Component models look like the ideal candidates to support these requirements - especially:

- reflexive components models w.r.t. observability and dynamic reconfiguration,
- hierarchical (or recursive) component models w.r.t. scalability, i.e. the uniform management of large scale distributed systems at arbitrary levels of abstraction - typically through the concept of *management domains* (possibly overlapping),
- open/extensible component models w.r.t the great variability in terms of components types, components life cycle, programming languages, etc. that has to be supported in large scale autonomic distributed systems.

Rationale: Advanced component models generally comes with advanced languages and tools such as Architecture Description Languages (ADLs), packaging and deployment models and management frameworks, powerful monitoring and dynamic reconfiguration support, QoS contracts support, etc. which are very valuable and (arguably) required mechanisms in autonomic systems. In the M2M use case, all scenarios require configuration, deployment and management capabilities as provided by components as a basis to almost all desired properties listed in the previous paragraph e.g. configurability and manageability through reflexive components, scalability through hierarchical components, reliability and availability through the intrinsic isolation provided by component-based architectures.

Type: Operational required mechanism

Dependency: RQ.MSM2M.Observability, RQ.MSM2M.Manageability

Assessment: The requirement assessment would be qualitative and reported in D5.4. The assessment of this requirements mainly boils down to the asses ement of RQ.Multi ServicesM2M.Observability and RQ.MultiServicesM2M.Manageability.

Target: W2 components

RequirementID: RQ.MSM2M.AutonomicDecision

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.LogOverlaySelfConfiguration,
UC.MSM2M.DeploymentOverlaySelfConfiguration,
UC.MSM2M.GracefulServiceDegradation,
UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing
UC.MSM2M.QualityOfContextManagement

Priority: Must

Description: We consider that an autonomic system is composed of an *autonomic infrastructure* superimposed on a *target (component-based) system*. The autonomic infrastructure is responsible for implementing a *control loop*, i.e. instrumenting the components of the target system for monitoring, detecting and notifying events, diagnosing the system based on these events, and making decisions to determine what and how corrective actions need to be executed, and finally executing the corrective actions on the components of the target system.An autonomic control loop conceptually should allow for advanced observation, diagnosis, decision making and reconfiguration (not to mention events and actions transport mechanisms we do not detail here).

Decision making refers to formalisms, languages a nd/or more operational mechanisms that allow for the specification and execution of *heuristics* and *policies*. Although complex mechanisms coming from artificial intelligence such as neural networks, bayesian networks, etc. can be used ; we consider simpler mechanisms such as Policy languages (e.g. Ponder or PDL), *deductive rules* (e.g. JBoss Rules) or *active rules* (or Event-Condition-Action rules) are good candidates to implement the core reactive behaviour of an autonomic control loop. Even simpler, decision making can be explicitely programmed in specific components.

Rationale: Decision making is a core functionnality of an autonomic control loop (it is the element that actually "closes the loop").

Type: Operational required mechanism

Dependency: RQ.MSM2M.Observability, RQ.MSM2M.Manageability

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check support of RQ.MSM2M.Observability
- check support of RQ.MSM2M.Manageability

- check availability of (at least one) decision making mechanism in Selfman middleware
- check usability of decision making in at least one autonomic use case

Target: W2 components, WP4 self-management services: decision making

RequirementID: RQ.MSM2M.AutonomicSimulation

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.LogOverlaySelfConfiguration,
UC.MSM2M.DeploymentOverlaySelfConfiguration,
UC.MSM2M.GracefulServiceDegradation,
UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing

Priority: May

Description: We consider that an autonomic system is composed of an *autonomic infrastructure* superimposed on a *target (component-based) system*. The autonomic infrastructure is responsible for implementing a *control loop*, i.e. instrumenting the components of the target system for monitoring, detecting and notifying events, diagnosing the system based on these events, and making decisions to determine what and how corrective actions need to be executed, and finally executing the corrective actions on the components of the target system. An autonomic control loop conceptually should allow for advanced observation, diagnosis, decision making and reconfiguration (not to mention events and actions transport mechanisms we do not detail here).

A simulation mechanism would allow for checking that a given reconfiguration decision would be actually correct and fruitful. As for typical artificial intelligence techniques, the idea is to generate possible action plans and to evaluate them, using modeling and simulation in our context.

Rationale: The M2M application particularly suits queuing network-based modeling and simulation. For modeling purpose, we need to characterize the performance and resource usage of M2M nodes. Since M2M nodes embed arbitrary business code with no assumption about its performance and resource usage, we will adopt either (1) an online characterization approach with a real workload and probes for observation purpose, or, (2) even better and whenever possible, an offline characterization approach with a self-regulated traffic generator sending messages to an M2M node, still including probes for observation.

Type: Operational optional mechanism

Dependency: none

Assessment: The requirement assessment would be qualitative and reported in D5.4. It boils down to checking the availability of a simulation mechanism in Selfman middleware.

Target: WP4 self-management services: simulation

RequirementID: RQ.MSM2M.TransactionalReconfiguration

Use case(s): UC.MSM2M.NodeFaultPrevention,
UC.MSM2M.NodeFailureRecovery,
UC.MSM2M.QoSManagement,
UC.MSM2M.ResourceConsumptionOptimization,
UC.MSM2M.LogOverlaySelfConfiguration,
UC.MSM2M.GracefulServiceDegradation,
UC.MSM2M.AvailabilityFromRoutingClusteringLoadBalancing

Priority: Should

Description: Reconfiguration transactions, which are sequences of reconfiguration actions (e.g. add, remove, replace components and bindings between components), have to be atomic, consistent, isolated and durable. Reconfiguration transactions can run concurrently so reconfiguration transactions have to be isolated. Consistency in this context is based on integrity constraints that can be generic (typing constraints, there cannot be cycles in the graph of components hierarchical containment) or application specific. Durability is based on persistent logging used for recovery.

Rationale: Transactional reconfiguration is a classical requirement towards ACID transactions for concurrency and recovery purposes. The need for transactional reconfiguration appears in all autonomic scenarios of the M2M use case.

Type: Operational mechanism

Dependency: RQ.MultiServicesM2M.Manageability (transactional reconfiguration needs reconfiguration mechanism)

Assessment: The requirement assesment would be qualitative and reported in D5.4:

- check support of RQ.MSM2M.Observability
- check support of RQ.MSM2M.Manageability
- check availability of transactional components reconfiguration mechanism in Selfman middleware
- check usability of transactional components reconfiguration mechanism in at least one autonomic use case (typically UC.MSM2M.NodeFailureRecovery). Typical tests will be of two sorts: fault injection by means of non correct reconfigurations so as to assess recovery and concurrent reconfigurations so as to assess concurrency control.

Target: W2 components

RequirementID: RQ.MSM2M.TransactionalMultiNodeProcessing

Use case(s): ?? si pas utile dans nos use cases, ajouter un use case exhibant le besoin ou virer le requirement ?

Priority: Should (the M2M application should support support this requirement)

Description: An M2M system is made of a set of nodes organized in an almost arbitrary graphs. A transactional data processing behaviour is sometimes required to encapsulate in one transaction a seuquence of processing on multiple nodes. There might be room here for non conventional transaction models here. For instance compensating transactions (as in SAGAs) is often preferable to classic flat transactions in an M2M context. There might be also room for nested transactions.

Rationale: There are several use cases for multi-node transaction processing in the M2M application: for instance the processing of fire alarm data from the sensors (smoke detectors) to the fire department through the M2M network of nodes should be transactional.

Type: Operational mechanism

Dependency: none

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check availability of a transactional mechanism in Selfman middleware
- check usability of transactional mechanism in M2M application in at least one autonomic use case

Target: WP3 self-manaing storage and transactions

RequirementID: RQ.MSM2M.TransactionalIntraNodeProcessing

Use case(s): UC.MSM2M.NodeFailureRecovery

Priority: May (NB: As a matter of fact, the functionnality is already supported by France Telecom M2M platform but in an ad-hoc fashion, i.e. without use of 'real transactions').

Description: An M2M node is typically made of 3 sub-components (in turn each made of sub-components): one that handles data reception (possibly from multiple connections), one that actually processes the data, and one that handles data emission. Due to the no data loss requirement, a transactional behaviour (especially atomicity) is required between these 3 sub-components inside a node component.

Rationale: There are several use cases for multi-node transaction proc essing in the M2M application: for instance the processing of fire alarm data from the sensors (smoke detectors) to the fire department through the M2M network of nodes should be transactional.

Type: Operational mechanism

Dependency: none

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check availability of a transactional mechanism in Selfman middleware
- check usability of transactional mechanism in M2M application in at least one autonomic use case

Target: WP3 self-manaing storage and transactions

RequirementID: RQ.MSM2M.TransactionalLogging

Use case(s): UC.MSM2M.LogOverlaySelfConfiguration

Priority: Should

Description: Reconfiguration transactions, which are sequences of reconfiguration actions (e.g. add, remove, replace components and bindings between components), have to be atomic, consistent, isolated and durable. Reconfiguration transactions can run concurrently so reconfiguration transactions have to be isolated. Consistency in this context is based on integrity constraints that can be generic (typing constraints, there cannot be cycles in the graph of components hierarchical containment) or application specific. Durability is based on persistent logging used for recovery.

Rationale: Transactional reconfiguration is a classic requirement towards ACID transactions for concurrency and recovery purposes. The need for transactional reconfiguration appears in all autonomic scenarios of the M2M use case.

Type: Operational mechanism

Dependency: RQ.MultiServicesM2M.Manageability (transactional reconfiguration needs reconfiguration mechanism)

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check support of RQ.MSM2M.Observability
- check support of RQ.MSM2M.Manageability
- check availability of transactional components reconfiguration mechanism in Selfman middleware
- check usability of transactional components reconfiguration mechanism in at least one autonomic use case (typically UC.MSM2M.NodeFailureRecovery)

Target: W2 components, WP3 self-manaing storage and transactions

RequirementID: RQ.MSM2M.TransactionalDeploymentOverlay

Use case(s): UC.MSM2M.DeploymentOverlaySelfConfiguration

Priority: May

Description: Transactional deployment refers to the management o f configuration activities by transactions (the prominent desired property is a tomicity). Concerned deployment activities are delivery, installation, con figuration, enactment (start), upgrade and updates, uninstallation (remov al) of software. As logging, deployment could be implemented as an overlay network which leads to transactions in overlay networks.

Rationale: In the M2M application, a transac tional deployment (initial installation and upgrades) of software inside domestic environments would be a very valuable asset.

Type: Operational mechanism

Dependency: RQ.MultiServicesM2M.Manageability

Assessment: The requirement assessment would be qualitative and reported in D5.4:

- check support of RQ.MSM2M.Observability
- check support of RQ.MSM2M.Manageability
- check availability of transactional components reconfiguration mechanism in Selfman middleware
- check usability of transactional components reconfiguration mechanism in at least one autonomic use case (typically UC.MSM2M.NodeFailureRecovery) typically by testing recovery following erroneous deployements (i.e. deployments that would build erroneous configurations).

Target: W2 components

RequirementID: RQ.MSM2M.ReputationManagement

Use case(s): UC.MSM2M.QualityOfContextManagement

Priority: May.

Description: This desired property refers to the ability for the system to measure the evolution of the quality of context (QoC) information processed by its nodes. This QoC is generally captured by the management of a reputation which qualifies the confidence one may have in the context information processed by a particular node.

Rationale: In the M2M context, monitoring the QoC level (by performing a thorough confidence assessment), and providing QoC guarantees become major requirements for contextual adaptations of safety-critical applications such as fire alarm detection, or temperature regulation, in which error-prone/malicious interpretation (e.g., false positive/false negative alerts) of unreliable context data can result in disaster situations.

Type: Operational mechanism

Dependency: RQ.MSM2M.Observability, RQ.MSM2M.Manageability.

Assessment: A reputation model must be chosen to measure and capture the evolution of the confidence of context information processed by an M2M node. The precise QoC dimensions capturing this confidence-level must also be defined (e.g., authenticity, confidentiality, integrity, precision, or privacy of context information). Metrics must also be chosen to measure the resilence of the system to QoC violations. Scenarios should be defined describing the injection of false (e.g., inaccurate, or with low QoC) context information in the system and measure its resilience. A succesful report of the requirement would be that in these various configurations, the system exhibits a sufficient level of resilience to QoC violations, by being able to reconfigure itself to select M2M nodes with the best reputation according to the selected chosen model.

Target: WP5 M2M middleware, WP4 self-protection.

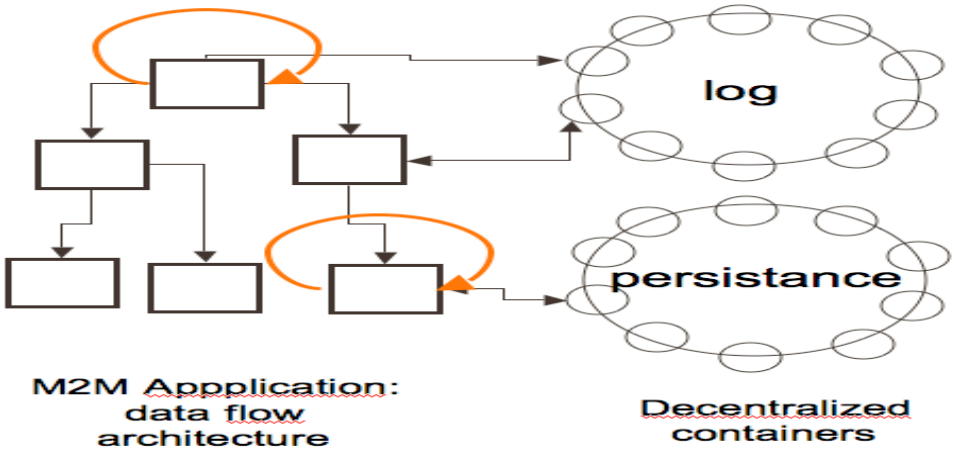


Figure 29: Co-existence of components and overlays: components for M2M functional part, overlays for non functional parts.

3.2.4 Conclusion

This section introduced a multi-service M2M (Machine-To-Machine) application and some associated autonomic (mis)use cases and requirements. Altough this work was driven by the M2M applications, we believe many use cases and requirements are common with other large scale autonomic distributed systems e.g. the 2 other applications described in the sequel on this deliverable.

As a general architectural feedback based on the M2M user requirements, we can observe that the architectural vision of M2M systems that underlies the M2M use cases (cf. Figure 29) in previous sections consists in introducing a SON-based architecture for handling non functional properties of the M2M application (with the typical example of the logging service in use cases) within a classical data flow (pipes and filters) component-based architecture (e.g. with explicit bindings) for the M2M core application itself. We can note that this vision in which components *coexist* with overlays might lead the concept of decentralized application server (more precisely *decentralized containers*) in which each infrastructure service (logging, persistance, security, etc.) is supported as individual/specific overlay. This architecture is already very innovative compared to current operational M2M systems.

Now, a even more innovative architecture we can envision from the study of the M2M use case and interactions with other Selfman WPs is a really disruptive architecture for M2M systems where the M2M system is itself implemented as an

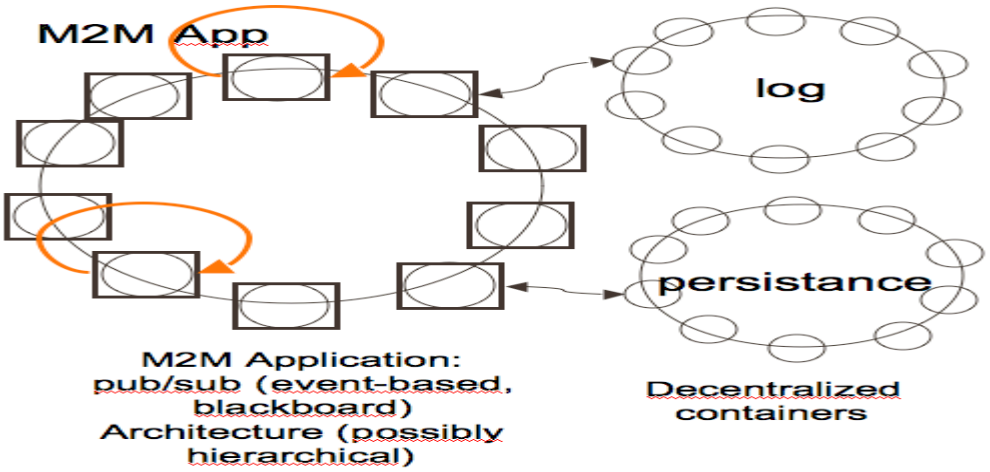


Figure 30: Integration of components and overlays: component-based overlay for M2M functional part - and possibly additional overlays for non functional parts.

event-based component-based overlay (cf. Figure 30). Indeed, the event-based or reactive component model that is being defined in Selfman is very close from an M2M infrastructure with a blackboard architecture rather than a explicit data flow architecture. Whether a M2M system can be modelled as a blackboard architecture based on SONs remains to be assessed and would need further research.

Complementary to this section, this deliverable contains in the appendices the following material produced by WP5/M2M Use Case:

- "Towards SELFMAN Security Misuse Cases: Looking at M2M Applications" introduces security issues in the M2M application from which the misuse case detailed in the body of the document (MUC.MSM2M.QualityOfContext) was taken out.

3.3 Distributed Database Use Case

This Section will give a short overview of the wiki(pedia) scenario that illustrates a distributed database system and discusses some of the requirements in the context of the SELFMAN project.

3.3.1 Applicative Context

Wikipedia is a free encyclopedia where any user can edit existing or add new content. The current version is a traditional three-tier application with proxies, application servers and database servers [5].

For SELFMAN, it can be treated as database-like application with versioning, replication and transactions. Trust management and self-optimization can also easily be integrated.

The wiki scenario covers the major topics of SELFMAN and in addition the real data sets are available for testing (http://download.wikipedia.org/backup-index.html) ranging in size from several kilobytes to several gigabytes.

3.3.2 Scenarios

Centralized Environment The architecture as of today follows a traditional three-tier design, where each server serves only a single purpose – database, business logic, or proxy (see Fig 31). The performance of the business logic servers and the proxies can be easily increased by adding more machines as this code is inherently parallel with almost no synchronisation overhead.

The shared state is stored in the database servers. Read as well as write accesses usually require synchronization between the nodes which limits the scalability.

Trusted Environment Instead of using separate hosts as proxies, application and database servers, each wikipedia server could run one peer of a structured overlay with a database like abstraction which provides the same functionality as the existing software.

The self-* components could autonomously balance the load and adapt system parameters to optimize the performance, e.g. the replication factor. Replica placement is also an issue as the performance can be improved by placing German content on servers in the center of Europe.

Untrusted Environment The server farm could be replaced by a structured overlay network where the peers run on the computers of the users of wikipedia. In this case self-* components, replication and trust management become more challenging. ...

Running the system in an untrusted environment means that the system has to handle a higher failure rate among nodes. Replication is needed to guarantee availability of the content in case of node failures. However with replication in SONs, keeping replicas in a consistent state becomes more complicated. In fact it is not possible to optimize for high availability, consistency and network partition

[5] http://meta.wikimedia.org/wiki/Wikimedia_servers

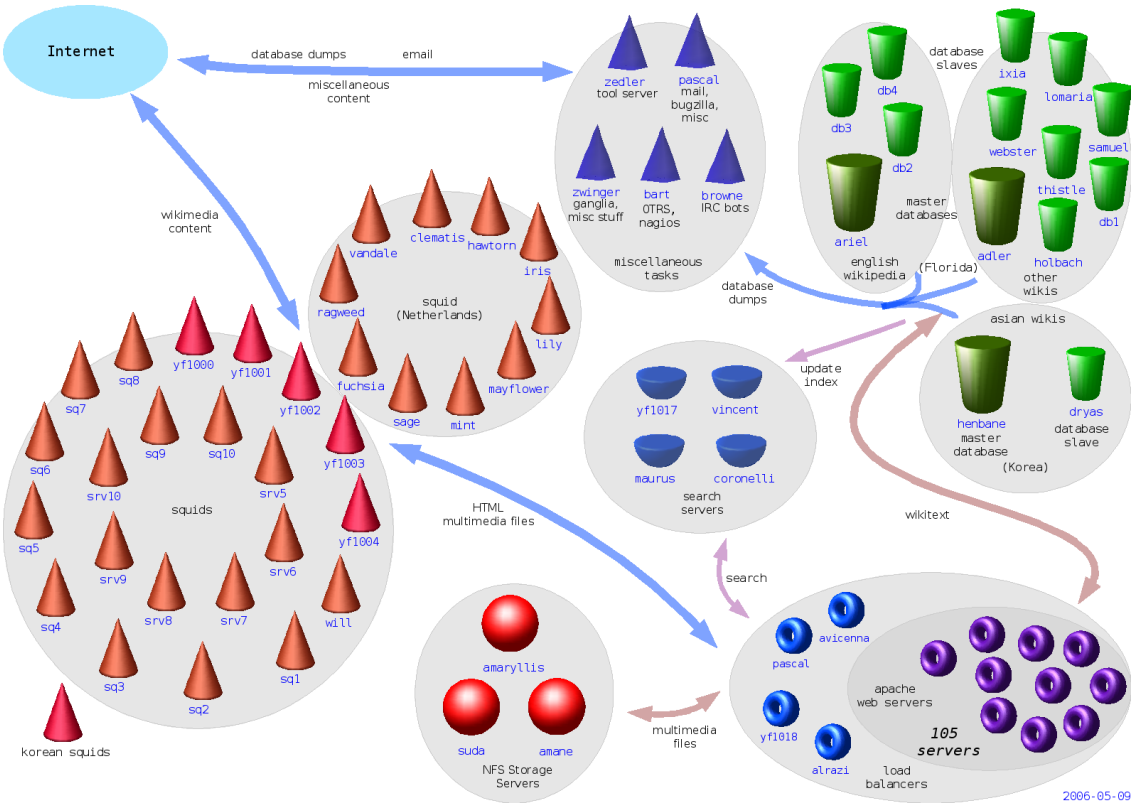


Figure 31: Wikipedia Server Farm as of 2006-05-09.

tolerance at the same time. It has to be investigated to which extent availability should be optimized and which consistency level is sufficient for a wiki scenario.

To provide strong consistency, update operations have to be atomic. We might even do transactions on data. Suppose a user editing some content. The system has to check whether the user had been working on the latest version at the time he wants to make his changes permanent. It should be ensured that changes made in between don't get lost. Data therefore gets some kind of version tags, and transactional mechanisms ensure that changes made by other users are not overwritten.

Besides regular editing and reading of pages, modern Wikis provide a host of additional features: support for attachments, user-based access control, storing per-page metadata (author, geographic coordinates etc.), anti-spam filtering, and several techniques to enhance navigation. These include fulltext searching, backlinks, and page categorization. All navigational features require storing of additional information that must be kept synchronized with page content. For example, to provide backlinks, it is necessary to have an additional reverse index that allows to find all links pointing to a page. Keeping these additional index structures synchronized requires the use of atomic transactions. Without atomicity, it becomes impossible to update page content and index structures consistently.

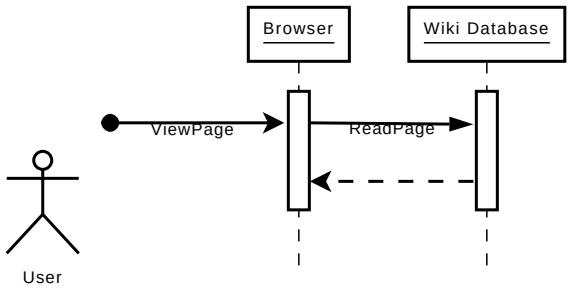


Figure 32: Timeline diagram for reading a page.

Use Case: UC.WikiDB.ReadPage

Scope: WikiDB, Selfman middleware: overlays

Description: The user employs a web browser to display the latest version of a wiki page. The returned version has to be consistent with the state of the database at the point the request was made.

Primary actor: Users

Stakeholders:

Preconditions: None.

Trigger: A user wants to read a page and uses his web browser to access the Wiki.

Basic course of event (cf. Figures 32):

1. Determine URL of page to be read
2. Read contents of the page

Alternative path: If no version of the requested page exists in the database, a notification is displayed instead of the content of the page.

Postconditions:

Miscellaneous:

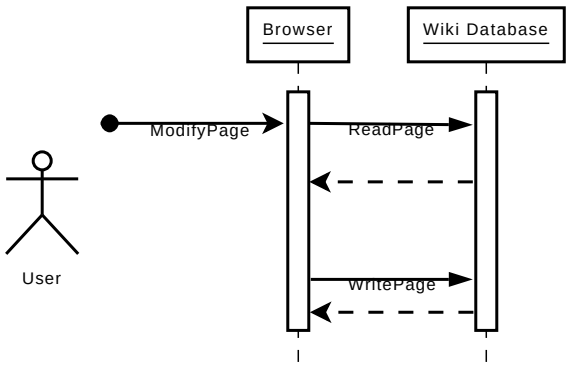


Figure 33: Timeline diagram for modifying a page.

Use Case: UC.WikiDB.ModifyPage

Scope:

Description: Editing a single page of wikipedia already requires a simple transaction. When updating a wikipedia page it is first read, before changes are applied on the read content. When the changes should be submitted it has to be checked whether the read version is still the current one and whether the changes can be applied. Thereby the version number of the read wikipedia page is compared with the version number of the page stored in the storage system. Additionally an update to a wikipedia page might require updating another page at the same time, e.g. a category page which is related to the particular page. In this case the update on the one page should only take place if it can be done on the other page at the same time. A transaction provides a mechanism to ensure this.

Relation name	Key	Value
CONTENTS	PageName and Version	Contents and Categories
CATEGORIESINDEX	CategoryName	PageName

Table 3: Simple relational model for storing wiki pages and categories

In a very simple model each page has a *name*, a *version number* and belongs to several *categories* (Tab. 3.3.2). To update a page, the current version number of the page in the database has to match the version number of the page the user was editing – there were no updates of the page since the user started to change the page. If check passed the page can be updated in the database and the categories table is updated to reflect the changes.

Primary actor: Users

Stakeholders: None

Preconditions: None

```
updatePage(PageName, Contents, LastVersionSeen)
  BOT //Begin Of Transaction
    if(CurrentVersion(PageName) == LastVersionSeen)
      OldContents = UpdateContent(PageName, Contents)

      OldCategories = ExtractCategories(OldContents)
      NewCategories = ExtractCategories(Contents)

      foreach c in NewCategories - OldCategories
        AddPageToCategory(PageName, c)

      foreach c in OldCategories - NewCategories
        RemovePageFromCategory(PageName, c)
    else
      FailTransaction
    end
  EOT //End Of Transaction
```

Figure 34: Code for updating a page

Trigger: A user wants to update a page and uses his web browser to access the Wiki.

Basic course of event (cf. Figures 33):

1. Determine URL of page to be changed
2. Read old content of the page
3. Modify content
4. Submit new content to the database

Alternative path: On step 4, if the database detects a concurrent modification, the submission will fail and the user has to go back to step 2.

On step 2, if the database detects that no previous version exists, the submission will create a new page.

Postconditions: All subsequent requests for the changed webpage will return the new version until further modifications happen.

Miscellaneous:

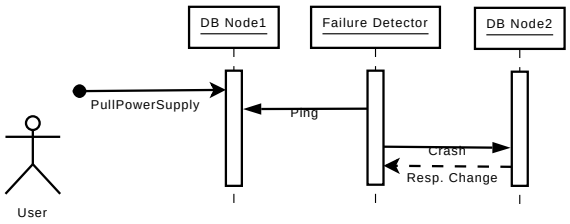


Figure 35: Timeline diagram for detection and replacement of a dying node.

Use Case: UC.WikiDB.NodeDies

Scope: WikiDB, Selfman middleware: overlays

Description: In large-scale distributed systems node failures are not the exception but a common event. Therefore the database has to be able to handle failures and still guarantee data consistency and availability.

Primary actor: Environment

Stakeholders:

Preconditions: None.

Trigger: A node can die for various reasons like e.g. hardware errors, software bugs or mis-configuration. The procedure for detecting and repairing the system is independent of these circumstances. Nodes have failure detectors that observe other nodes in the system. If a failure detector suspects a node as failed it will create a notification as shown in figure 35.

Basic course of event (cf. Figures 36):

1. A DB node dies because of e.g. a software error, power or network outage. The failure of the node is detected by some remaining nodes in the system.
2. As soon as nodes suspect a node to have failed they will update the structure of the distributed DB. Pointers to the failed node will be replaced with pointers to alive nodes in the system.
3. When a node fails another node in the system has to become responsible for the failed node's items. Thus it will have to create copies of the data this node stored to ensure availability of data.
4. As the new responsible node might be overloaded with data upon it created new copies, the load balancing mechanism has to redistribute the data.

Alternative path: On step 4 no node has too much load due to creating new copies. In that case no load balancing is necessary.

Postconditions: When a node dies the capacity in terms of storage and performance is reduced proportionally to the capacity of that node. However neither the availability nor the consistency of the data is affected.

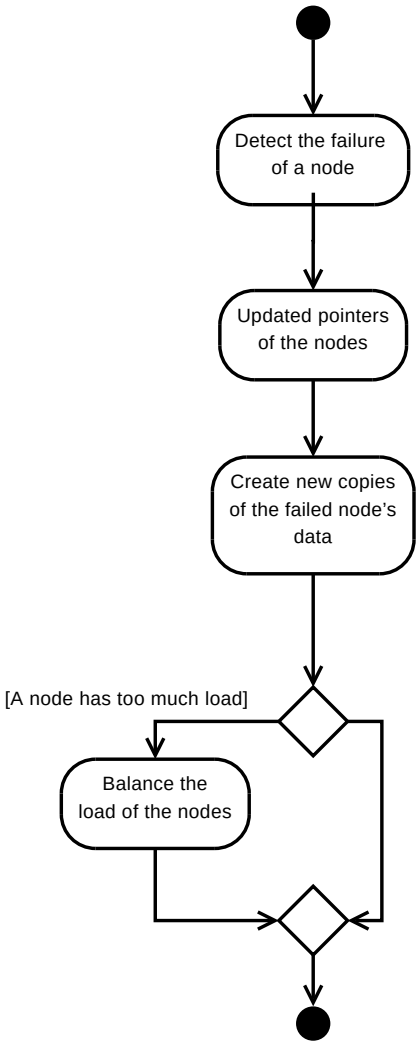


Figure 36: Activity diagram for the handling of a node failure.

Miscellaneous:

Use Case: UC.WikiDB.AddNode

Scope: WikiDB, Selfman middleware: overlays

Description: In large-scale distributed systems new nodes can be added to the system in order to increase the capacity of the system. If a new node is added the structure of the distributed DB should be adjusted accordingly. The new node has to get data it is responsible for and the load in the system should be equally balanced to all nodes if necessary.

Primary actor: Environment

Stakeholders:

Preconditions: None.

Trigger: A new node can be added to the system whenever the capacity and performance of the system should be increased.

Basic course of event (cf. Figures 37):

1. A DB node is added to the system.
2. As soon as nodes in the system know about the new node they will update the structure of the distributed DB. Pointers to the new node will be created.
3. The new node has to fetch the data it is responsible for.
4. As the new node introduces more capacity in the system the load can be redistributed in order to decrease the load from overloaded systems.

Alternative path: On step 4 no node has too much load due. In that case no load balancing is necessary.

Postconditions: When a node is added the capacity in terms of storage and performance is increased proportionally to the capacity of that node. Availability and consistency has to be maintained while adding the new node.

Miscellaneous:

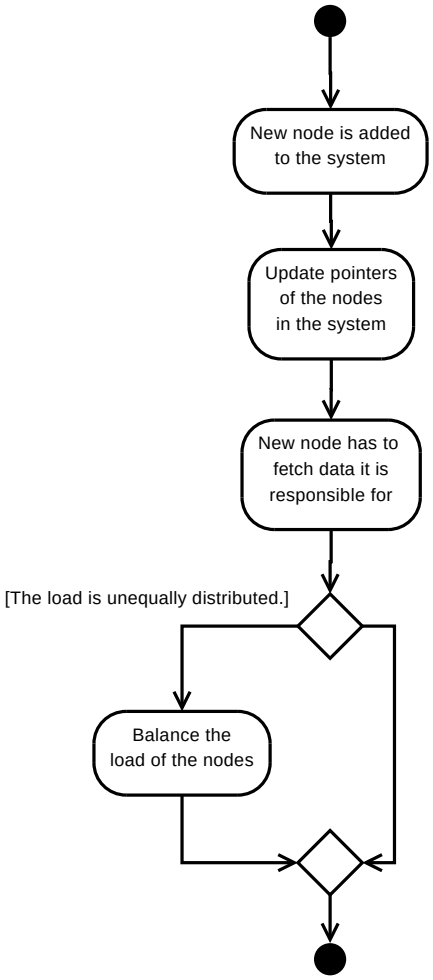


Figure 37: Activity diagram for the handling of a node failure.

Searching for category pages In order to retrieve consistent results, searching wikipedia pages requires the use of transaction processing, too. A range query is necessary to query the overlay for all pages that fall in a specific category (Fig. 38).

If it is required that no concurrently added page should be missed by this query, predicate locking is necessary. Besides read-locking the index entries themselves, in this simple case, this requires locking the whole category for insertion and deletion.

3.3.3 Autonomic Scenarios

Self-Optimization: Global Load-Balancing Research on SONs usually assumes that all stored entries are equally popular. Real-world access patterns usually follow a Zipf-distribution, where a few items are very popular and the majority of the items are seldom accessed. In addition the access pattern varies over the day

```
categoryPages(categoryName)
  BOT //Begin Of Transaction
    pages = GetMatching("CategoriesIndex", "CategoryName=" + categoryName)
    Sort(pages, "Version")
    return pages
  EOT //End Of Transaction
```

Figure 38: Code for searching category pages

because of the different timezones.

A self-optimizing component is needed which adjusts the mapping of the key space of the SON to the nodes and adjusts the replication factor for individual keys.

Self-Healing: Recovery of Data on Failure This scenario describes the mechanisms needed to handle unexpected node failures. First of all, the integrity of the data stored on the failed node has to be guaranteed. Using redundancy, the database has to implement safety measures to be able to restore the missing data. Using failure detectors neighboring nodes in the overlay will monitor each other and find replacement nodes in cases of failure.

3.3.4 Requirements

This section identifies the requirements and components associated with the scenarios described above. The distributed scenario needs three components:

- Distributed Database
- Distributed Webserver
- Distributed Trust Management

Each of them and the specific requirements are described in the following.

Components

Distributed Database The current backend of wikipedia is a MySQL database distributed over 15 servers where the distribution of data over the server is configured by hand and certain tasks can only be performed by a few and not by all machines (Master-Slave).

A distributed database build on top of a DHT must at least provide the following functionality.

Versioning. Each article is stored with its whole history and there exists a total order over all the versions of an article.

At any time, changes to an article can be reverted and the differences between several versions can be displayed.

When a user updates an article it has to be guaranteed that the changes were made on the latest version, i.e. the user saw the latest version before he started to change the article.

Distributed Webserver A distributed webserver which accesses the database and renders the web pages.

Distributed Trust Management A distributed trust management component to assess the quality of the contributions of users.

Autonomics

Self-Repairing To increase the reliability and availability of the database all information has to be replicated over several nodes to tolerate the failure of small numbers of nodes and repair mechanism are needed to fix failed nodes autonomously.

Self-Optimization The structured overlay has to employ an autonomous load-balancing scheme which takes popularity of items and the influence of time-zones into account.

Transactions A transaction mechanism is needed to ensure atomicity of the required operations for an update. It is also needed to prevent concurrent update from violating the consistency of the content.

Overlay networks There very few requirements to the overlay network which go beyond the features provided by standard SONs. Of importance is only the maintenance overhead in the Trusted Environment scenario. As the failure is very low the ring maintenance should be adapted accordingly.

3.3.5 Conclusion

The main challenge is to build a system based on a SON, which provides the user with an acceptable performance and sufficient data consistency. Therefore a proper trade-off between availability, data consistency and network partition tolerance has to be found.

3.4 P2P TV Application

This section introduces the P2PTV use case proposed par the PeerTV company. The goal of this use case is to be able to distribute live media-streams from any source to large number of consumers (nodes), 100 K to Million nodes, without using any expensive central resources.

3.4.1 Applicative Context

IP multicast is currently mostly disabled by most ISPs due to the extra cost incurred on the routers and incompatibilities between different Autonomous Systems (AS?s). The current approach for live streaming uses large and expensive server equipment allocated nearby the router devices to perform media distribution. An alternative solution is to use overlay networks of consumer nodes to broadcast live-streams, thereby minimizing the cost of deployment and provisioning by individual media providers. On the top of the overlay the nodes will be dynamically organized as multicast trees. Nodes will cooperate in streaming by exploiting their upload bandwidth capacities to multicast streams to other nodes.

3.4.2 Use Cases

Peer-to-peer live streaming is challenging problem in the context of Selfman as one needs to:

- maximize the total utilization of upload bandwidth,
- minimize latency, and to
- dynamically reconfigure the trees during network dynamism.

The first requirement stresses that the solution needs to ensure that the actual available upload bandwidth at each node should be utilized as much as possible. Any solution must, therefore, adapt to the given upload bandwidth of the individual nodes. This implies that even nodes with petty upload bandwidth should be utilized.

The second requirement puts focus on latencies between the actual nodes. It also implies that the depth of the multicast trees should be shallow to minimize latencies (this is at least true given identical node latencies).

Finally, the solution should continuously reconfigure the system, as there will always be some network dynamism. By network dynamism we refer to i) nodes joining/leaving/failing, or ii) network capacity changing due to network congestion etc.

3.4.3 Requirements

There are a number of requirements on the systems built in Selfman to be able to handle this type of applications:

1. It should be possible to dynamically build multicast trees on the top of the overlays designed in the project.

2. It should be possible to optimize continuously the nodes in the multicast trees so that nodes with higher upload capacities are on the top of the multicast trees and nodes with lower capacities near to the leaves.

3. It should be possible to optimize continuously the nodes in the multicast trees so that the latency between neighboring nodes in the overlays are minimized

4. It should be possible to optimize continuously the nodes in the multicast trees so that the traffic between AS?s are minimized

5. It should be possible to upgrade the software of the overlays remotely and dynsmically. All these requirements pose interesting challenges on the results of Selfman.

3.4.4 Conclusion

Dynamic live-streaming is an interesting application scenario for Selfman as it requires most of the self* properties that the project is set to design on the top of overlay networks.

4 Conclusion on T5.1/D5.1

The purpose of the WP5 is to provide use cases and requirements in different applicative contexts (applications) so to help the Selfman project as a whole to choose which application(s) will be demonstrated, and then ultimately to perform evaluations of the Selfman demonstrators and more globally the Selfman technonologies and overall approach (components + overlays).

The Selfman DOW (Description of Work) elaborated at the beginning of the project was not very explicit, precise concerning target Selfman application area(s). It only mentions 'application servers' as the broad applicative context and occasionnaly 'J2EE application' server and 'WWW server'. As stated by Wikipedia: "Although the term application server applies to all platforms, it has become heavily identified with the Sun Microsystems J2EE platform; however, it has also come to encompass servers of Web-based applications, such as integrated platforms for e-commerce, content management systems, and Web-site builders." Also, the E-Plus industrial partner was excepted to provide user requirements and trace data about WWW servers - while the other industrial partner at that time, France Telecom, was expected to contribute on WP2 and WP4 on component-based self-* mechanisms and performance evaluations in WP5. Selfman reacts to the unexpected departure of the project of E-Plus by:

- elaborating 2 use cases in replacement: the wiki application server proposed by ZIB and the M2M apllication server proposed by France Telecom,
- and investigating 2 addtional use cases with 2 potential new industrial partners: the P2P TV application from Staak (ex P2PT V) investigated with KTH and the J2EE application server from Bull investigated with France Telecom and INRIA.

The result of the analysis of the 4 candidate applications is given in table 4.

Applications	Self-*Properties	Components	Overlays	Transactions
Multi Service M2M	Strong	Strong	Medium	Medium
Wiki Distributed Storage	Strong	Low	Strong	Strong
P2P TV	Strong	Low	Strong	No
J2EE Application Server	Strong	Strong	No	Low

Table 4: Synthesis of operational requirements of the 4 Selfman applications on WP1-Overlays, WP2-Components, WP3-Transactions and WP4-Self-* Services.

As a synyhesis of the analysis, the following observations can be done:

- All 4 applications exhibit strong requirements towards autonomic properties from a general point of view, i.e. without considering how the self-* properties would actually be implemented (how they could use components and overlays).
- The J2EE application exhibits strong requirements in term of management and self-management based on components . It exhibits low requirements

toward transactions (except transactional reconfiguration) and no requirement at all, at this moment, towards overlay networks. In our discussions with Bull, it appeared that Bull has indeed a growing interest for more large scale distributed architectures for their current clustered application server ; but that their technical vision on the subject was currently not matured enough engage Bull in a collaborative project. More specifically, they currently have no idea on how overlays could be used in J2EE context.

- The P2P TV application currently exhibits low requirements on components and no requiments at all on transactions. Altogether, this application is intrinsically overlay-oriented and might be a good support to demonstrate direct self-optimization features (video QoS management).

- the wiki distributed batabase and M2M application appear more complex (e.g. multi service aspect in the M2M application), innovative (P2P video streaming has already been investigated elsewhere) and complementary: the former is intrinsically overlay-oriented while the later is intrinsically component-oriented.

Apart from the dificulties and additional work generated in WP5 by the E-Plus leave of the project and consecutive discussions with Staak and Bull, a major difficulty experienced in T5.1 was the lack, in the project consortium, of a clear, explicite and shared architectural vision on the combination of components and overlays and how this would facilitate the construction and deployment of autonomic features in distributed systems - which is at the very heart of Selfman. This surely is a challenge for the second year of the project.

Depending on the choice of Selfman demonstrator(s), the following points could/should be deepered:

- the introduction of components in overlay networks supporting the Wiki and P2P video streaming applications - for instance by a component-based implementation of peers forming the overlay,

- the introduction of overlay networks in the M2M application - for instance by implementing the logging service as an overlay. We can note that this vision in which components *coexist* with overlays might lead the concept of decentralized application server (more precisely *decentralized containers*) in whih each infrastructure service (logging, persistance, security, etc.) is supported as individual/specific overlay. Another and perhaps more disruptive architecture for M2M systems that would be very interesting to investigate is the one mentionned in the conclusion of the M2M Section 3.2.4 where the M2M system is itself implemented as an component-based overlay (hence the architecture style moves from data flow to blackboard).

5 Appendices: publications and other documents produced by T5.1

Appendices of the M2M use case contains the following documents:

- "Towards SELFMAN Security Misuse Cases: Looking at M2M Applications" introduces security issues in the M2M application from which the misuse case detailed in the body of the document (MUC.MSM2M.QualityOfContext) was taken out.

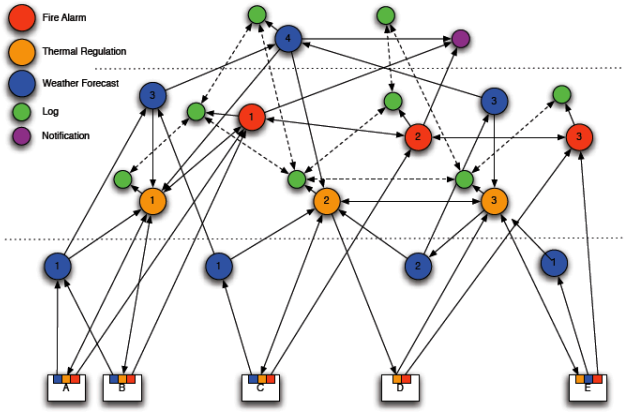
Towards SELFMAN Security Misuse Cases: Looking at M2M Applications

Orange Labs

Marc Lacoste

France Telecom Research & Development
SELFMAN meeting, Grenoble, November 21th, 2007.

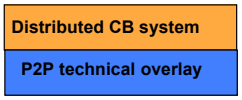
M2M Applications: The Big Picture



Two Possible Architectures for M2M Applications

THE ADVENTURES OF SELFMAN

- Distributed CB system over P2P technical overlay network.



- Closer to "real" M2M systems.
- *Focus:* data routing and processing.
- "Fixed" application-level topology explicitly defined by an ADL description.
- Non functional services (e.g., logging) = self-organized overlay.

- P2P application-level overlay over P2P technical overlay.

- More innovative/disruptive.
- Two overlay networks.
- Application-level network implemented as a P2P overlay.

Deployment Environment for M2M Applications: Some Assumptions

THE ADVENTURES OF SELFMAN

- Deployment on an open vs. closed (VPNs) infrastructure.
- Machines can host both M2M and P2P overlay nodes.
- Sensors and M2M services are uniquely identified.
  - Unique IDs (MAC address-based) provided by TTPs.
  - Alternative schemes for identification/authentication of nodes and information between nodes.
- Logged sensor data may be confidential (accountability).
  - Mechanisms for privacy management?
  - Different views/roles/authorizations in the P2P distributed database.

Attacks on the M2M Applicative Layer

- *Flooding M2M machines:*
  - Inserting malicious/garbage data in the system at different levels: network protocols (injection of malicious traffic), middleware, and M2M applicative layer.
- *DDoS at the middleware level:*
  - Continuously inserting/removing a large number of sensors resulting in M2M node layer instability.
- *Sensor spoofing at the M2M service layer:*
  - Inserting a malicious (with fake identity) sensor node that sends corrupt data and/or floods the rest of the M2M system.
- *M2M node spoofing (MITM attack):*
  - Inserting a malicious (with fake identity) M2M node that corrupts, stops data routing (e.g., critical packets are dropped), and/or floods the rest of the M2M system.
- *Important requirement:* guarantee the authenticity of the context information exchanged between M2M nodes, used to trigger critical adaptations (thermal regulation, fire alarm).
  - False positive or false negative situations could result in critical disasters.
  - An infrastructure is needed to ensure the "**quality of context**" (QoC): confidentiality, integrity, authenticity, privacy, precision, etc.

Attacks on the P2P Technical Layer

- *P2P node spoofing:*
  - Inserting malicious nodes in the P2P overlay that fake logging, log corrupted data, or flood communications between overlay nodes.
- *P2P management disruptions:* attacks on the overlay management (DHT structure, topology management).
  - Routing algorithms (eclipse attacks).
  - Sybil attacks.
- *Fairness violation and selfish behaviors:* overlay nodes "don't play by the rules" (free riding):
  - Reading/but not writing in the logs.
  - Dispatch logging tasks into the logs to other nodes.
  - Not very relevant for M2M systems but might be considered for other applications (Distributed wiki, P2P TV).