There are three parts of our functionality that cause this behavior (i.e. table contains LOB data grows very large):

1. Upon rolling back an insertion of LOB data, we do not deallocate the LOB pages. Instead, the LOB pages remain allocated, and they are potentially used the next time LOB data is inserted.

2. When inserting a LOB value, if the value is larger than one page, we break it up into page-sized fragments, with the last fragment being the remainder (i.e. possibly smaller than a page).

3. When inserting a LOB value (or fragment), we typically search through the existing LOB pages to find free space for it. However, we have a performance optimization. If the value or fragment is exactly the size of one page, we do not search through the existing LOB pages. Instead, we simply allocate a new page for it. The reason for this optimization is that typically the free space on the existing LOB pages are all less than one page in size. This is because the most common way to get free space on a LOB page is when we insert a LOB value or fragment less than one page in size, leaving the remaining space on that page as free. Although we can get an entire free LOB page from rolling back a LOB insertion, this is more uncommon. Thus, rather than searching through the existing LOB pages and most likely not finding an entire free page, we just allocate a new page.

So, when a user rolls back a LOB insertion, we leave all the LOB pages allocated. The next time they try to insert LOB data, if the LOB data is greater than a page, we allocate new pages for it, only reusing the existing pages for the remainder. If this is also rolled back, we have a net increase in empty LOB pages.

This behavior is by design. When it does not result in optimal behavior in some cases, we can use an easy workaround to reduce the table size:

First, you should run DBCC CLEANTABLE. This will go through the LOB pages and deallocate any free ones. After running this, the next time they insert LOB data greater than a page, instead of allocating a page at the end of the file, we allocate one of the ones we deallocated in DBCC CLEANTABLE.

That by itself should be enough to stop your database from growing. However, ideally you would also run ALTER INDEX ... REORGANIZE WITH (LOB\_COMPACTION = ON). This will compact all of the LOB pages, freeing up extents and reserved pages. Or alternatively, you can run DBCC SHRINKDATABASE. This will first compact the LOB pages like ALTER INDEX and then actually shrink the database file.

1. In actuality, you would probably want to run DBCC CLEANTABLE and then DBCC SHRINKDATABASE once to shrink your database down to a manageable size.
2. Then after that, you would just periodically run DBCC CLEANTABLE and ALTER INDEX to maintain that size.

<http://technet.microsoft.com/en-us/library/ms143729.aspx>

<http://technet.microsoft.com/en-us/library/ms143729(v=sql.105).aspx>

<http://technet.microsoft.com/en-us/library/ms143729(v=sql.100).aspx>

<http://technet.microsoft.com/en-us/library/ms143729(v=sql.90).aspx>